# Accelerated Deconvolution of Radio Interferometric Images Using Orthogonal Matching Pursuit and Graphics Hardware [1]

Jonathan Van Belle, James Gain, Richard Armstrong



[1]The Financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

**Abstract**

Deconvolution of native radio interferometric images constitutes a major computational component of the radio astronomy imaging process. An efficient and robust deconvolution operation is essential for reconstruction of the true sky signal from measured correlator data.

Traditionally, radio astronomers have mostly used the CLEAN algorithm, and variants thereof. However, the techniques of compressed sensing provide a mathematically rigorous framework within which deconvolution of radio interferometric images can be implemented.

We present an accelerated implementation of the orthogonal matching pursuit (OMP) algorithm (a compressed sensing method) that makes use of graphics processing unit (GPU) hardware, and show significant accuracy improvements over the standard CLEAN.

In particular, we show that OMP correctly identifies more sources than CLEAN, identifying up to 82% of the sources in 100 test images, while CLEAN only identifies up to 61% of the sources. In addition, the residual after source extraction is 2.7 times lower for OMP than for CLEAN. Furthermore, the GPU implementation of OMP performs around 23 times faster than a 4-core CPU.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Listings

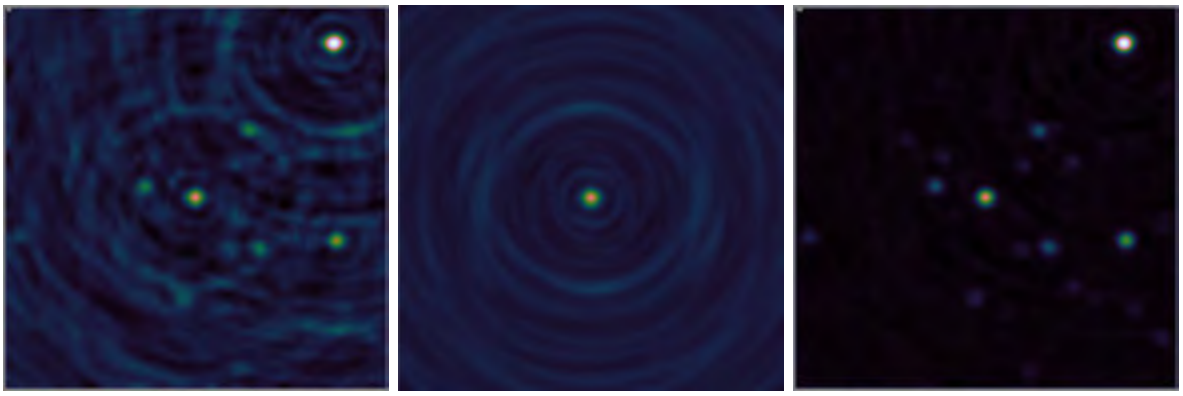
# Chapter 1

# Introduction

## 1.1 Radio Interferometry

Radio interferometry is the observation electromagnetic radiation at radio wavelengths by using an array of receivers. The use of multiple receivers allows maps to be provided with a resolution nearly equal to that possible with a single large receiver that encompasses all of the smaller receivers. This can mean an effective maximum resolving area ranging from kilometres (such as the VLA) to thousands of kilometres (for Very Long Baseline Interferometry). This area only corresponds to the *resolution* of a large receiver, and not to the amount of electromagnetic radiation gathered. As such, the sensitivity of an interferometer only scales with the sum of the actual effective area of each component.

Radio interferometers measure the sky through the *spatial coherence function* instead of sampling the radio sky directly. Thus, when the field size is restricted around the direction of interest and the antennas are coplanar enough, reconstruction can be approximated to a square and flat plane patch of the sky. In this case, they measure points in the 2-dimensional Fourier domain; these points are called *visibilities*. These points then need to be transformed by the



a b c

Figure 1.1.1: Deconvolution of an image: (a) a dirty image, (b) the corresponding PSF, and (c) the corresponding deconvolved image (which has been convolved with a restoring beam).

2-dimensional Fourier transform in order to obtain an image of the sky.

The points an interferometer measure correspond to a spatial frequency of the sky which depends on the physical separation of the receivers and the direction of observation. Since it is usually infeasible to have telescopes at every possible baseline, the Fourier domain is therefore sub-sampled.

## 1.2 Deconvolution

Because of this sub-sampling, there are multiple possible images that match the measurements. The simplest approach to imaging (that is, producing an image) is to consider all unmeasured visibilities as zero. This results in what is known as the *dirty image*, and suffers from artefacts, as can be seen in Figure 1.1.1a. Because of the assumption of zero values, the sky's image is effectively convolved with a *Point Spread Function* (PSF), which is the result of the interferometer measuring a point source at the phase centre.

Most deconvolution algorithms attempt to remove this convolution by subtracting the PSF from where the sources are thought to lie (typically the brightest points in the dirty image).

*Deconvolution* is the process of selecting the image that is the closest to the actual sky from among the possible images. Figure 1.1.1c shows a deconvolved image using the same measurements. This image is much sparser, and is much closer to the expected appearance of the sky. It is also much easier to identify the sources in Figure 1.1.1c than in Figure 1.1.1a.

From our knowledge of the radio sky, many images will consist of only a few small or point sources on a background of a low-intensity noise, resulting in a sparse image. As such, a good approach to deconvolution is to find a sparse image that matches the measurements.

## 1.3 Compressed Sensing

Nyquist-Shannon sampling theory requires that there be at least twice as many measurements as points in the image in order to perfectly reconstruct it. We can, however, use prior knowledge of our image (that it is sparse in a pixel dictionary) to reduce the number of measurements required.

The theory of *Compressed Sensing* [Candès, 2006] states that, if the measurements can be calculated as a linear combination of the signal, we have a high probability of being able to reconstruct a $d$-dimensional $s$-sparse signal (a signal with only $s$ non-zero values) or *$s$-weakly sparse* signal (a signal where s values are dominant, with the other values close to zero) with as few as $slogd$ measurements, provided the linear coefficients for each measurement are nearly orthogonal to each other (the inner product of the coefficients of different measurements are close to zero) [Kunis and Rauhut, 2008].

In terms of radio interferometry, the measurements are points on the 2-dimensional Fourier plane. As such, the coefficients are columns of the 2-dimensional Fourier transform matrix. Fortunately, these satisfy the orthogonality constraint [Candès, 2006]. This means that there is a high probability of reconstructing the image of the actual sky from interferometric measurements

so long as the image is sparse enough (has less than half as many non-zero pixels as the number of measurements).

Orthogonal Matching Pursuit (OMP) is a compressed sensing algorithm that optimises this process by recursively selecting the single pixel that would most minimise the difference image (the difference between the transform of the image and the measurements), and then weighting all the selected pixels in the image so that they best approximate the measurements. Schwardt [2012] has found that this strategy produces superior results (in terms of sparsity, contrast, and execution time) compared to other popular compressed sensing techniques.

## 1.4 Computation

The computational cost of measuring the sky with an interferometer increases quadratically with the number of radio antennas. As such, it is important for each process involved with generating the image to execute as quickly as possible. A schematic overview of these processes is shown in Figure 2.0.1. This is especially true of the Square Kilometre Array (SKA), a future radio interferometer, which will have one to three orders of magnitude more antennas of any other interferometer.

Since the OMP algorithm is generalised for any linear transform, one can accelerate the process by specialising it for image deconvolution. This includes the use of Fast Fourier Transforms (FFTs) to accelerate matrix-vector multiplications, and simplification of some matrix-matrix multiplications.

Furthermore, since most of the operations in OMP involve matrix operations and Fourier transforms, the algorithm should provide a significant speed-up when implemented on graphics hardware, which is efficient at computing such calculations.

In addition, if the preceding algorithm (gridding) and subsequent algorithms (source detection, image analysis) are already implemented on GPUs, implementing OMP on GPUs might allow for fewer transfers between GPU memory and system memory. This has the potential to create a significant performance improvement, since this transfer can cause delays given the relatively low bandwidth available between the GPU's memory and the CPU's memory.

## 1.5 Objective

For this research, we intended to implement an OMP algorithm for radio interferometric deconvolution. This implementation was then compared against CASA's CLEAN, a well established existing deconvolution implementation. This comparison included a test on the number of sources successfully extracted, the number of sources that were not extracted, the number of spurious sources, the relative brightness of extracted sources, the relative position of extracted sources, and the remaining residue after source extraction.

In addition, the algorithm was implemented on the GPU. The runtime of this algorithm was then compared to that of the CPU implementation.

## 1.6 Findings

The OMP algorithm implemented in this work significantly outperformed CLEAN at source-finding, matching up to 82% of the sources modelled in the 100 synthetic sky models used in our tests, while CLEAN only matched up to 61% of the sources. Furthermore, the residual images for OMP were, on average, 2.9 times lower in intensity than those of CLEAN.

It was initially found that OMP generated many more spurious sources (false positives) than CLEAN. However, by modifying the source-matching algorithm, it was found that most of these spurious sources do, in fact, correspond to actual sources, or can be predicted based on intensity and location. After implementing this modification, it is shown that OMP only produces a few more spurious sources than CLEAN.

Additionally, our GPU implementation achieved up to a 83 times speed-up over the single-threaded CPU implementation, and a 23 times speed-up over the 4-threaded implementation.

## 1.7 Thesis Structure

The theory of interferometric imaging and compressed sensing is discussed in much greater detail in Chapter 2. In particular, we look at the radio interferometry pipeline (Section 2.1); sampling theory, including compressed sensing (Section 2.2) and deconvolution (Section 2.3).

Chapter 3 provides a basic understanding of *General Purpose Graphics Processing Unit* (GPGPU) computing.

Chapter 2.4 provides a brief overview of papers directly relating to this thesis.

In Chapter 4, we examine each major step required in OMP, and analyse the resulting algorithmic complexity.

We will detail the specifics of implementing OMP on both the CPU and on the GPU in Chapter 5. We also look at how to perform the required matrix inversion.

Chapter 6 details the results produced by the OMP implementation. We compare the resulting sources extracted from a synthetic sky to those extracted after deconvolution by CLEAN.

We present our conclusions in Chapter 7.

# Chapter 2

# Background

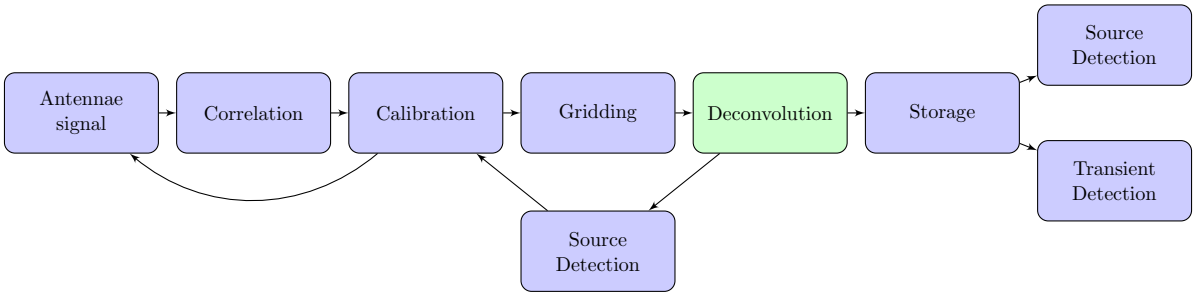



Figure 2.0.1: The radio interferometry pipeline. The deconvolution step is highlighted.

This chapter will provide a brief overview of the theory required to understand why deconvolution is necessary and where it fits in the radio astronomy pipeline. A basic diagram of how the major steps in interferometry interact is shown in Figure 2.0.1.

The antennas signal is the raw output from an antenna, which is amplified and run through a correlator to compare the signal to that of a different antenna. After some assumptions and calculations, this results in the sampled visibilities (points on the 2-dimensional Fourier plane). This process is detailed in Section 2.1.1.

*Calibration* attempts to remove distortion due to the the effect occurring and accumulating on the line of sight (e.g. atmosphere fluctuation at high frequencies, or ionosphere at low frequencies) as well as noise from the system (e.g. receiver noise, gain, and direction-dependent beam shape). Some calibration approaches are sky-model dependant, which require parsing the signal through most of the pipeline several times, modifying the signal processing based on the detected sources. This is detailed in Section 2.1.3.

In order to use a Fast Fourier Transform (FFT) instead of the slower Discrete Fourier Transform (DFT), the measured visibilities have to be placed onto a regular grid. This process is known as *gridding* and is detailed in Section 2.1.4. Gridding provides us with a dirty image and a PSF, allowing us to start *deconvolution*.

Compressed Sensing (CS) techniques allow an image to be recovered under certain constraints with far fewer measurements than was required by the prior Nyquist-Shannon Sampling

Theorem. Section 2.2 details these constraints and provides several CS techniques, including Orthogonal Matching Pursuit.

Section 2.3 applies this theory to deconvolution. We look at two methods previously used for deconvolution, CLEAN and MEM, and consider that a method developed based on Compressed Sensing theory might produce superior results.

The final deconvolved image is typically stored in an archive, and is thus ready for analysis, such as source extraction or transient detection.

## 2.1 Radio Interferometry

The intent of radio astronomy is to observe the sky in the radio spectrum in order to enhance our understanding of the Universe. As such, we want to make our measurements as accurate as possible (within computational constraints).

Interferometers can be used to provide us with higher resolution images than the individual antennas can provide alone.

However, interferometers do not directly sample the sky; instead, after some computation and simplifying assumptions, they sample the 2-dimensional Fourier space.

### 2.1.1 Interferometer Measurements

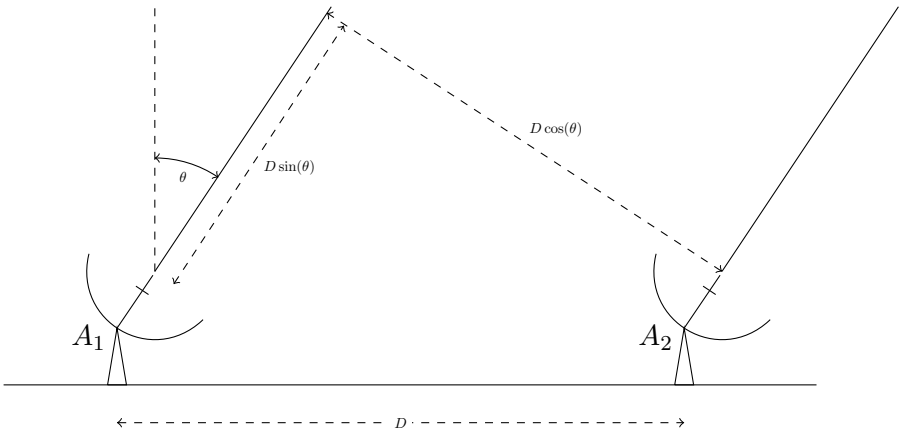



Figure 2.1.1: A simple 2-element interferometer with baseline $D$. The antennas are pointed at a patch of sky at an angle $\theta$ away from the zenith.

Consider a distant point source observed by an interferometer as described by Figure 2.1.1.

If the wave front from the point source reaches antenna $A_2$ at a time $t$, it will reach antenna $A_1$ at a time $t + \tau_g$ where $\tau_g = \frac{D}{c}\sin\theta$ is called the geometric delay.

The signals from the antennas pass through amplifiers, which include filters to select the required frequency, $\nu$. This output is then combined in a *correlator*, which is a voltage multiplier followed by a time averaging circuit. Thus, if the output signals from the amplifiers are $V_1(t)$ and $V_2(t)$, then the correlator output is given by $\langle V_1(t)V_2(t)\rangle$, where angular brackets denote a time average.

For any frequency $\nu$ of the receiver bandwidth, we can write:

$$V_1(t) = V\cos\left(2\pi\nu t - \tau_g)\right)$$

$$V_2(t) = V\cos\left(2\pi\nu t\right).$$

Thus the output of the correlator is:

$$\begin{aligned} r &= V^2\cos\left(2\pi\nu\tau_g\right) \\ &= \left(\frac{V^2}{2}\right)\left(\cos(4\pi\nu t - 2\pi\nu\tau_g) + \cos(2\pi\nu\tau_g)\right) \end{aligned}$$

By taking a time average $\Delta t \gg (4\pi\nu)^{-1}$, we remove the high-frequency term $\cos(4\pi\nu t - 2\pi\nu\tau_g)$ from the final output:

$$r = \left(\frac{V^2}{2}\right)\cos(2\pi\nu\frac{D}{c}\sin\theta) \tag{2.1}$$

We can now consider the response (of the 2-element interferometer shown in Figure 2.1.1) of an extended source.

Let $I(\hat{s})$ represent the sky intensity in the direction $\hat{s}$, let $A(\hat{s})$ denote the effective collecting area of the antennas in direction $\hat{s}$, and let $\vec{b}$ denote the vector from the position of antenna $A_1$ to the position of antenna $A_2$. We can consider the extended source as the integral over the entire surface of celestial sphere of the point source response described by Equation 2.1:

$$r_e = \int A(\hat{s})I(\hat{s})\cos\frac{2\pi\nu\vec{b}\cdot\hat{s}}{c}\ d\Omega\ .$$

Because of the even cosine function, this response is only sensitive to the even part of $I$. We address this by implementing a second, odd correlator which has a $\pi/2$ phase delay:

$$r_o = \int A(\hat{s})I(\hat{s})\sin\frac{2\pi\nu\vec{b}\cdot\hat{s}}{c}\ d\Omega\ .$$

We can then combine these into a single complex measurement, called the *complex visibility*, as:

$$V = Ae^{-i\phi}$$

with amplitude

$$A = \sqrt{{r_e}^2 + {r_o}^2}$$

and phase

$$\phi = \arctan\frac{r_o}{r_e}.$$

Thus the visibility measured by the two-element interferometer is

$$V = \int A(\hat{s}) I(\hat{s}) e^{-2\pi i \nu \vec{b} \cdot \hat{s}/c} d\Omega \ . \tag{2.2}$$

For implementation, this equation requires a coordinate system. The one usually chosen is the vector $(u, v, w)$, where $w$ points in the direction of interest. We can then choose $u$ and $v$ to point East and North, respectively. We further denote positions on a plane representing the sky with $(l, m, n)$ coordinates, where $l$ and $m$ are direction cosines measured with respect to the $u$ and $v$ axes and, since we confine our image space to a plane, $n$ then becomes $\sqrt{1 - l^2 - m^2}$. An example of these coordinates is shown in Figure 2.1.2.

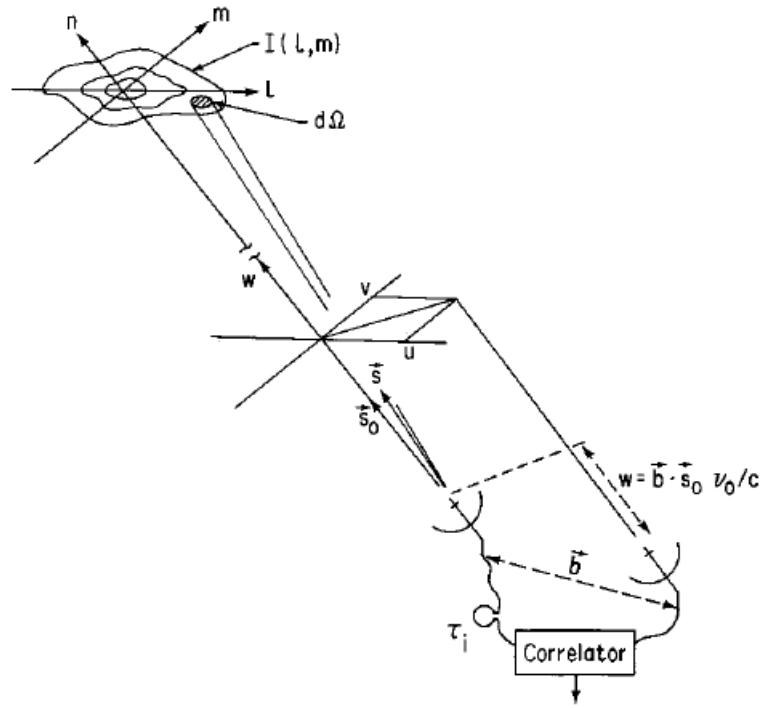



Figure 2.1.2: The $(u, v, w)$ and $(l, m, n)$ coordinates. From Synthesis Imaging in Radio Astronomy II [Taylor et al., 1999].

In these coordinates, we get the following terms:

$$\begin{aligned} \frac{\nu \vec{b} \cdot \hat{s}}{c} &= ul + vm + wn, \\ d\Omega = \frac{dldm}{n} &= \frac{dldm}{\sqrt{1 - l^2 - m^2}} \end{aligned}$$

Thus Equation 2.2 becomes:

$$V(u, v, w) = \iint A(l, m) I(l, m) e^{-2\pi i \left(ul + vm + w(\sqrt{1 - l^2 - m^2} - 1)\right)} \frac{dldm}{\sqrt{1 - l^2 - m^2}}. \tag{2.3}$$

In order to then obtain $I(l, m)$ from the visibilities, we need to invert this equation. If we

can reduce the equation into the form of a two-dimensional Fourier transform, we can use the inverse Fourier transform, which is well understood. We are able to do this for two cases.

The first is when all the baselines are coplanar to each other. Then, by choosing the $w$-axis such that $w = 0$, Equation 2.3 becomes:

$$V(u,v) = \iint A(l,m)I(l,m)e^{-2\pi i(ul+vm)}\frac{dldm}{\sqrt{1-l^2-m^2}}. \tag{2.4}$$

This is a two-dimensional Fourier transform, with the inverse:

$$\frac{A(l,m)I(l,m)}{\sqrt{1-l^2-m^2}} = \iint V(u,v)e^{2\pi i(ul+vm)}dudv. \tag{2.5}$$

In order for the baselines to remain co-planar as the earth rotates, antennas must lie along an East-West line. In this case, the $w$ axis will lie in the direction of the Earth's axis, and any measurement away from this axis will be hampered. This is shown in Figure 2.1.3 (note that the observed image in the $(l,m)$ coordinates will be increasingly distorted from the actual source at $(\alpha_0, \delta_0)$ as it moves away from the Earth's axis).

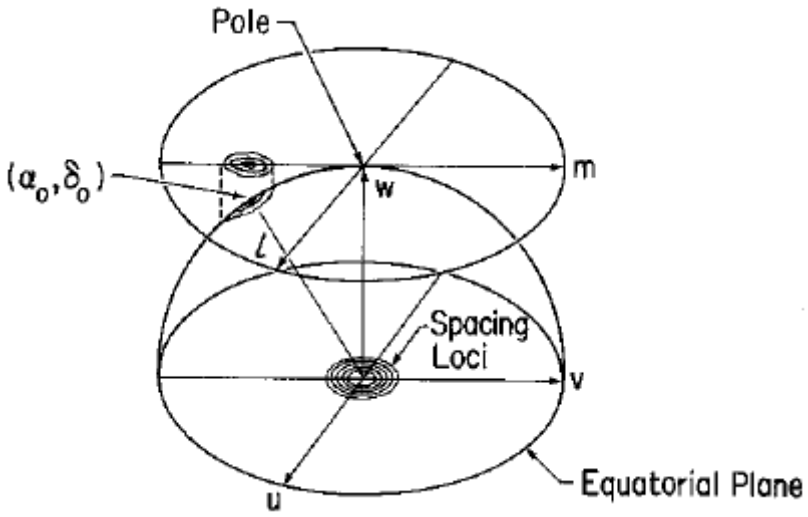



Figure 2.1.3: An example of an East-West interferometer array observing a source at $(\alpha_0, \delta_0)$. The spacing loci correspond to the antennas baselines. From Synthesis Imaging in Radio Astronomy II [Taylor et al., 1999].

In the case of observations away from the celestial poles, we need baselines parallel to the Earth's axis. If we restrict ourselves to a small region of the sky so that $|l|$ and $|m|$ are small enough such that

$$\sqrt{1-l^2-m^2} - 1 \approx -\frac{1}{2}(l^2+m^2)w \approx 0 \tag{2.6}$$

Equation 2.3 becomes the Fourier transform

$$V(u,v) = \iint A(l,m)I(l,m)e^{-2\pi i(ul+vm)}dldm. \tag{2.7}$$

And we obtain the inverse Fourier transform

$$A(l,m)I(l,m) = \iint V(u,v)e^{2\pi i(ul+vm)}dudv. \tag{2.8}$$

### 2.1.2 Visibility Sampling

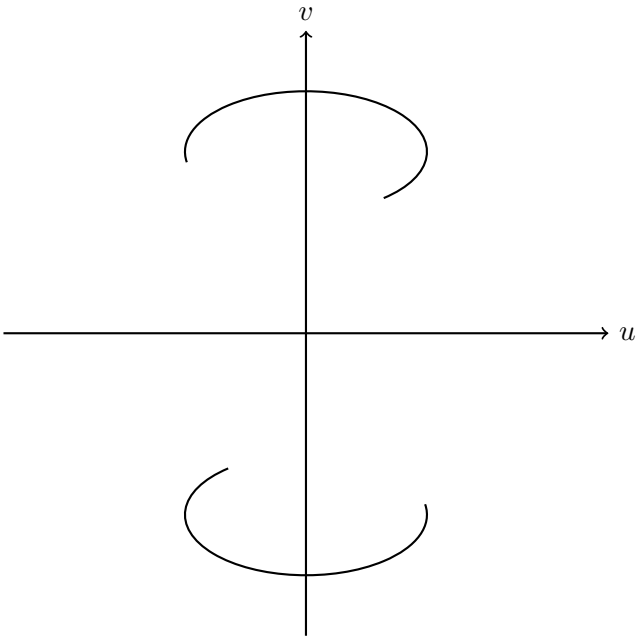



Figure 2.1.4: An example of a two-element interferometer sampled as it tracks across the sky. The lower curve corresponds to the negative baseline vector (and represents the complex conjugate). Adapted from Synthesis Imaging in Radio Astronomy II [Taylor et al., 1999].

In order to invert the Fourier transforms in Equations 2.5 and 2.8, we need to take the integral of $V(u,v)$ over the entire $(u,v)$-plane. However, we only have measurements for $V(u,v)$ corresponding to the baseline vectors of the physical antennas. While we can still approximate the Fourier inverse without the fully sampled $(u,v)$-plane, the approximation is better with more baseline vectors.

Since almost all of the radio sky is unchanged over the time of an observation (except by rotating with the earth), one way to increase the number of baselines is to observe the same region of the sky at a later time when the earth has rotated, and thus the antennas are in a different configuration as seen from the source. This will allow for one original baseline to cover an ellipse of baselines on the $(u,v)$-plane as the Earth performs a full rotation, as is shown in Figure 2.1.4. The missing section of the ellipse is the duration for which the direction of observation is obscured by the Earth).

Another method is to physically move some of the antennas, thus providing new baseline measurements involving those antennas (the VLA and ALMA are examples of reconfigurable interferometers).

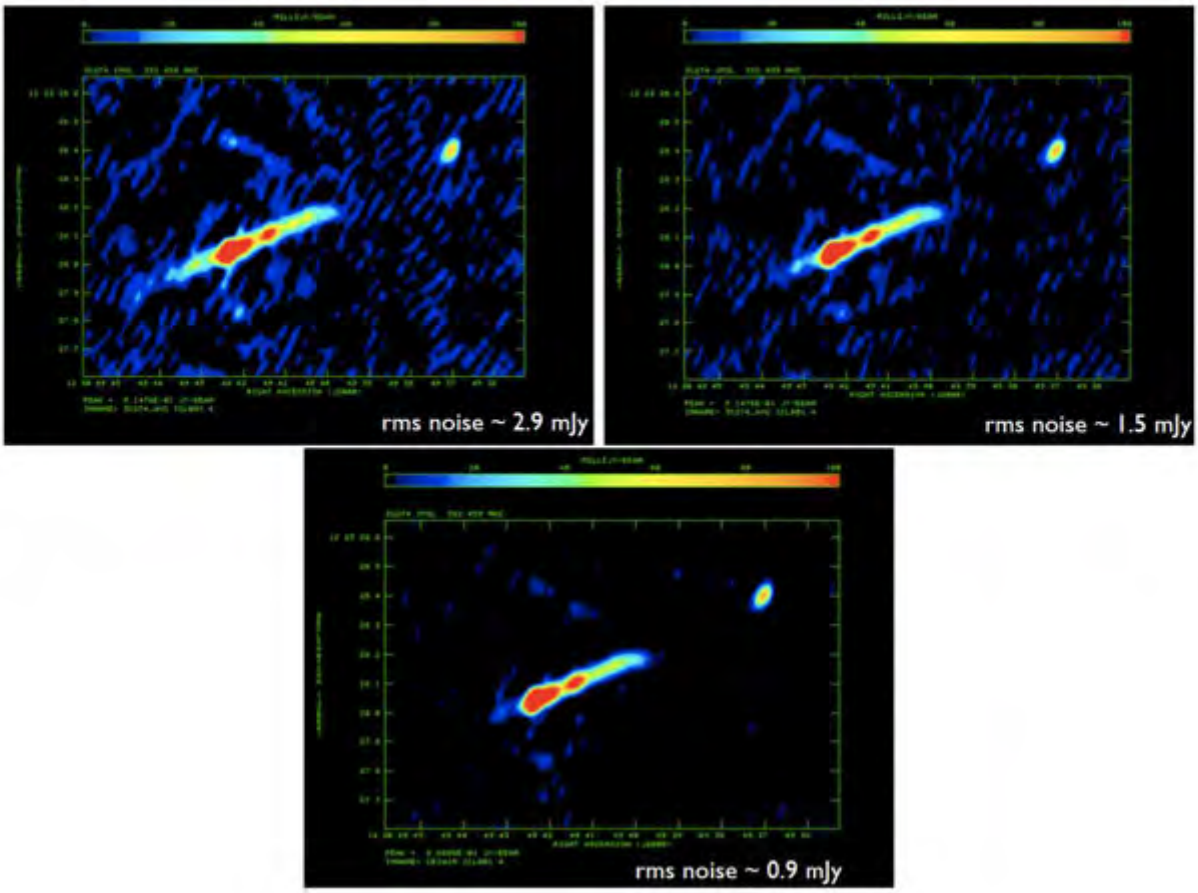



Figure 2.1.5: Example of self-calibration after the initial calibration (left), and after the first (right) and second (bottom) self-calibration iterations in a hybrid approach [Garrett, 2010].

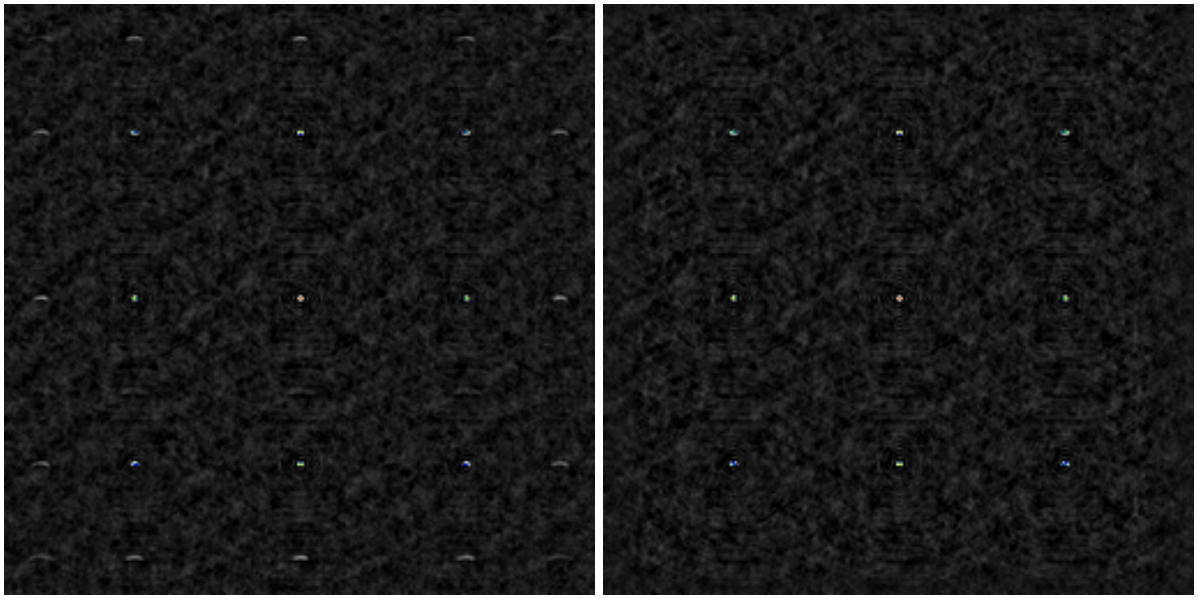
We define the *Sampling Function*, $S$, as $S(u, v) = 1$, if $V(u, v)$ has been measured, and $S(u, v) = 0$, otherwise.

### 2.1.3 Calibration

Atmospheric conditions can affect the propagation of electromagnetic radiation through the atmosphere, affecting each radio antenna differently. There are also differences in each antenna's beam pattern and receiver gain. *Calibration* is the process of measuring the error these factors introduce and accounting for them in our *instrument model*. This allows us to factor these errors out when imaging.

One method of calibration is to observe a known object and use the difference between what we know and what we observe to refine the instrument model. However, since some error contributions change over the duration of the observation, this approach can only go so far.

Another approach, called *self-calibration* is to use knowledge gained from the current observation. We produce an image using the current instrument model, identify the resolvable sources, and then use those sources to refine the instrument model. This refined instrument model can then be used to produce a better image for self-calibration. This process is repeated until the image no longer improves [Kazemi and Yatawatta, 2013].



a b

Figure 2.1.6: Example of aliasing (in the dirty image) due to poor gridding: (a) the result of matching each pixel to its nearest neighbour (b) the result of a convolution. The central 9 points are correct, while the additional sources around the edges of (a) are erroneous.

Typically, both approaches are used in what is called *hybrid mapping*. A known source is used for calibration to produce the first image, which is then used to start the self-calibration process. Figure 2.1.5 shows the improvement after the first and second self-calibration steps, with the image features becoming more apparent, and a reduction in the noise.

An important consideration is that the images used for self-calibration are deconvolved. Thus a more accurate deconvolution method would allow for a larger improvement in the instrument model in each step. This would result in fewer required self-calibration steps, or in a better final instrument model.

### 2.1.4 Gridding

Fourier transforms form the main computational component and cost in calculating Equations 2.5 and 2.8. As such, we would like to make use of the Fast Fourier Transform (FFT) algorithm. However FFTs require that the visibility values be regularly spaced.

In order to obtain this regular spacing, we form a regularly spaced grid, and determine the values of points on this grid by interpolation from nearby, measured visibilities.

The simplest interpolation, placing each visibility onto the nearest gridpoint, can result in aliasing [Thompson and Bracewell, 1974], which can seen in Figure 2.1.6. Note the additional sources present around the edges of image (a).

As such, some form of interpolation should be used. Bilinear interpolation (that is, linear weighting based on distance to nearby points) works, but still leads to some artefacts.

In particular, a smooth 2D function is chosen which has an arbitrary relationship with

the distance (such as a Gaussian, or a prolate spheroidal function), and use a convolutional interpolation with this function. This function, in this scope, is called the kernel function.

### 2.1.5 The Dirty Image

Due to most interferometers not sampling the entire visibility space, there are typically many unknown points. Since the FFT requires each point on the grid to have some fixed value, we need to assign a value to these unsampled visibilities.

The simplest solution is to ignore the contribution of the unknown visibilities, that is, to assume that they are all 0. This effectively multiplies the visibility with the sampling function, $S$, so that the predicted visibility, $\tilde{V}$, is:

$$\tilde{V}(u,v) = S(u,v)V(u,v)$$

Since this is a point-wise multiplication, we can apply the convolution theorem. This results in an image convolved in the Fourier transform of the sampling function.

We call the Fourier transform of the sampling function the *Point Spread Function* (PSF) or *Dirty Beam* ($B_D$). We call the image resulting from the convolution by the PSF the *Dirty Image* ($I_D$).

This dirty image is usually used as the starting point of deconvolution, in which we attempt to determine reasonable values for the unknown visibilities, thereby removing the sampling function effect in the image plane.

## 2.2 Sampling Theory

### 2.2.1 Nyquist-Shannon Sampling Theorem

The Nyquist-Shannon sampling theorem states that, "If a function $f(t)$ contains no frequencies higher than $n$Hz, it is completely determined by giving its ordinates at a series of points spaced $1/2n$ seconds apart" [Shannon, 1949].

This means that a signal can be perfectly reconstructed if it is uniformly sampled at at least twice its frequency.

Landau [1967] later proved that the points do not need to be sampled uniformly; in particular, only the signal's average sampling rate must be at least twice its frequency.

While the Nyquist-Shannon sampling theorem provides an upper bound on the samples required to reconstruct a general signal, we can often reconstruct a signal with far fewer samples when we know something about the signal we are reconstructing.

### 2.2.2 Compressed Sensing

Compressed Sensing (CS) theory attempts to accurately approximate compressible signals using fewer samples than the sampling theorem requires. In particular, if some signal, $x$, has a sparse representation in some orthonormal basis, we find that the $N$ most important coefficients could

be found with only $n = O(N \log(m))$ measurements, rather than the $2m$ required by the sampling theorem, where $m$ is the highest frequency contained in the signal [Candès, 2006].

Finally, a good approximation to these $N$ coefficients can be found through linear programming.

So, given a general linear measurement system:

$$\Psi x = y \tag{2.9}$$

where $x$ is a vector of the target signal we want to measure (of size $n$), $y$ a vector of the measurements (of size $m$), and $\Psi$ an $m \times n$ matrix, which is called the *measurement system*, if $m \ll n$ (there are more unknowns than observations), there are usually multiple solutions for $x$. We make use of the fact that $x$ is *sparse* in order to find a good solution.

### 2.2.3 Sparsity

Sparsity is required for the signal, $x$, to be compressible in some way. This means that there must be a representation of the signal in some space which has fewer entries than its natural representation.

A signal, $x$, is sparse (resp. weakly sparse) if there is a representation, $\alpha$, of $x$ which has fewer than $n$ elements. That is, $x$ can be represented by $x = \Phi\alpha$ for some $l \times n$ matrix, with $l$ less than $n$ and other elements are 0 (resp. close to 0). Thus Equation 2.9 becomes

$$\Psi\Phi\alpha = y \quad \text{or} \quad \Theta\alpha = y \tag{2.10}$$

where $\Theta = \Psi\Phi$. We call $\Phi$ the *sensing matrix*.

It is often the case that $x$ already has few non-zero entries, in which case $\Phi$ can be taken as the identity matrix (there is no change of space of representation in that case, because $x$ is already sparse in its natural representation space).

### 2.2.4 Incoherence

Mutual coherence between two matrices, $\mathbf{A}$ and $\mathbf{B}$, is a measure of the largest correlation between any two rows of $\mathbf{A}$ and $\mathbf{B}$, given by

$$\mu(\mathbf{A}, \mathbf{B}) = \max_{1 \leq k,j \leq n} \left| \langle \mathbf{A'}_k, \mathbf{B'}_j \rangle \right|,$$

where $\mathbf{A'}$ contains the rows of $\mathbf{A}$ normalised such that $\langle \mathbf{A'}_i, \mathbf{A'}_i \rangle = 1$.

Compressed sensing requires there to be incoherence between the measurement system, $\Psi$, and the sensing matrix, $\Phi$. In the case where the sensing system is the Fourier domain, and if $x$ is already sparse in its natural space, then $\Psi$ is assembled from selected rows of the Fourier transform matrix $\mathbf{F}$ and $\Phi$ is the identity matrix, and $\mu(\mathbf{F}, \Phi) = 1$.

Candes and Romberg [2007] show that a signal can be recovered with high probability when the number of measurements, $m$, satisfies

$$m \geq C\mu^2(\Psi, \Phi)s \log n,$$

for some constant $C$, where $s$ is the sparsity of the signal (the number of non-zero entries in its sparse representation).

Lustig et al. [2007] have found experimentally that, for incoherent sampling (that is, $\mu(\Psi, \Phi) = 1$), $m$ need only be 2-5 times larger than $s$ in order to have a high probability of reconstructing the signal.

### 2.2.5 Restricted Isometry Property

The previous theory focuses on a signal with a sparse representation. However, it is more often the case that most entries will have very small values instead of zero values. In order to apply CS theory to this case, we need to introduce the Restricted Isometry Property (RIP).

We first define the isometry constant $\delta_s$ for each $s \in \{1, 2, \ldots\}$ as the smallest number such that, for all $s$-sparse vectors $\alpha$,

$$(1 - \delta_s)\|\alpha\|_2{}^2 \leq \|\Theta\alpha\|_2{}^2 \leq (1 + \delta_s)\|\alpha\|_2{}^2.$$

We say that $\Theta$ satisfies the RIP of order $k$ if $\delta_k < 1$.

Candès [2006] shows that, if the isometry constant $\delta_{2s} < \sqrt{2}-1$, the $\ell_1$ minimisation solution, $\alpha^*$, of Equation 2.10 satisfies

$$\|\alpha^* - \alpha\|_2 \leq C\frac{\|\alpha - \alpha_s\|_1}{\sqrt{s}} \tag{2.11}$$

where $\alpha_s$ is $\alpha$ with all but the largest $s$ entries set to zero, thereby providing a bound on the reconstruction error.

In order to allow for noise in the measurement system, we modify Equation 2.10 to

$$\Theta\alpha + E = y$$

where $E$ represents the measurement error. Equation 2.11 now becomes

$$\|\alpha^* - \alpha\|_2 \leq C_0\frac{\|\alpha - \alpha_s\|_1}{\sqrt{s}} + C_1\epsilon \tag{2.12}$$

where $\epsilon$ represents a bound on the amount of noise.

Candès [2006] shows that Gaussian measurement matrices, binary measurement matrices, Fourier measurement matrices and incoherent measurement matrices all satisfy the RIP.

### 2.2.6 Basis Pursuit

The direct solution to reconstructing a sparse signal (in the absence of noise) is to solve

$$\min_{\alpha} \|\alpha\|_0 \quad \text{subject to} \quad \Theta\alpha = y$$

where $\|\alpha\|_0$ is equal to the number of non-zero entries in $\alpha$.

This is a combinatorial problem (NP Hard) and thus is computationally intractable, so a different approach should be taken.

By providing bounds on the reconstruction error, equation 2.11 shows us that the $\ell_1$ norm can provide a good solution (in particular, it provides an upper bound on the reconstruction error). This leads us to the Basis Pursuit (BP) algorithm by solving

$$\min_{\alpha} \|\alpha\|_1 \quad \text{subject to} \quad \Theta\alpha = y. \tag{2.13}$$

In the case of noise in the measurement system, we can use Equation 2.12 to obtain the Basis Pursuit De-Noise ($\text{BP}_\epsilon$) algorithm by solving

$$\min_{\alpha} \|\alpha\|_1 \quad \text{subject to} \quad \|\Theta\alpha - y\|_2 \leq \epsilon. \tag{2.14}$$

By recasting the problem to its unconstrained Lagrangian form, we get the Quadratic basis Pursuit ($\text{QP}_\lambda$) problem

$$\min_{\alpha} \left( \|\Theta\alpha - y\|_2{}^2 + \lambda \|\alpha\|_1 \right) \tag{2.15}$$

where $\lambda$ is a balancing factor between the signal and the noise.

The BP problem can be solved using linear programming, while the $\text{BP}_\epsilon$ and the $\text{QP}_\lambda$ problems can be solved as second-order cone programs. The $\text{QP}_\lambda$ problem can also be solved using quadratic programming.

### 2.2.7 Matching Pursuit

While $\ell_1$ optimisation techniques have guaranteed convergence, they often take a large number of iterations to achieve an acceptable solution. *Greedy* algorithms are designed to accurately reconstruct the signal significantly faster.

The Matching Pursuit (MP) algorithm, detailed in Algorithm 2.2.1, iteratively selects the best element in $\alpha$ which would minimise the residual ($r = y - \Theta\alpha$), and adds it to the solution vector. Since $\Theta$ satisfies the RIP, we can choose this element as the element of maximum absolute value in $\Theta^{\text{H}} r$.

### 2.2.8 Orthogonal Matching Pursuit

While Matching Pursuit does converge, in a given iteration the solution vector, $\alpha$, is not necessarily the optimal solution vector with respect to its non-zero components. The Orthogonal

**Algorithm 2.2.1** Matching Pursuit

$r \leftarrow y$
**while** $r$ is significant **do**
    $p \leftarrow \Theta^{\mathrm{H}} r$
    $i \leftarrow \arg\max |p_j|$
    $\alpha_i \leftarrow p_i$
    $r \leftarrow y - \Theta\alpha$

Matching Pursuit (OMP) algorithm, detailed in Algorithm 2.2.2, addresses this problem by re-balancing all previously selected values such that the solution vector is optimal for the selected components.

The OMP algorithm operates exactly like MP until the best element is selected. Because we will be weighting this component in each future iteration, its index is stored in an (initially empty) vector, $D$.

For each selected element, $\alpha_i$, its contribution to $y$ is determined by the $i^{\text{th}}$ row of $\Theta$. As such, the matrix $\Theta_D$ is set to be the matrix containing only the columns of $\Theta$ whose index is in the set $D$.

Since each selected element's contribution is perfectly represented by $\Theta_D$, the solution vector of $\arg\min \|y - \Theta_D\alpha\|_2$ will result in the minimum possible residue for the selected components. Finding this solution vector is a well known, solved problem in linear algebra (least-norm solution of underdetermined equations).

Because we have minimised the contribution from each of the selected components, the component selected in the next iteration will not be a result of poor prior weights, resulting in a better component in the next iteration.

**Algorithm 2.2.2** Orthogonal Matching Pursuit

$r \leftarrow y$
$D \leftarrow \emptyset$
**while** $r$ is significant **do**
    $p \leftarrow \Theta^{\mathrm{H}} r$
    $i \leftarrow \arg\max |p_j|$ where $j \notin D$
    $D \leftarrow D \cup \{i\}$
    $\alpha \leftarrow \arg\min \|y - \Theta_D\alpha\|_2$ ▷ Only the selected components $\alpha_j, j \in D$ are determined by this problem. Define $\alpha_j = 0$ for $j \notin D$.
    $r \leftarrow y - \Theta\alpha$

## 2.3 Deconvolution

Thanks to the convolution theorem, the effect of sub-sampling the Fourier plane (i.e. multiplying the Fourier plane by a masking function, which is literally the sampling function given by the observation) is a convolution of the actual sky image with the PSF (see Section 2.1.5). Deconvolution is the attempt to remove this convolution.

Since an unsampled area in the Fourier plane could take any value, there are infinite possible solutions to the deconvolution equation. Thus what we want is to obtain the most reasonable values for the unsampled visibilities.

In this section we describe two of the most commonly used deconvolution algorithms, CLEAN and MEM. We also describe some of the common CLEAN variants.

Finally, we discuss why a deconvolution method based on CS theory could be effective.

### 2.3.1 CLEAN

The CLEAN algorithm [Högbom, 1974] provides a solution to the convolution equation by modelling the sky as a number of point sources on a 0-intensity background. The proposed algorithm[1] is as follows:

**Algorithm 2.3.1** Högbom CLEAN

1: $\hat{I} \leftarrow 0$
2: **while** $I^D$'s maximum is above some threshold **do**
3: $\quad p, q \leftarrow \arg\max I_{p,q}$
4: $\quad I^D \leftarrow I^D - PSF * \gamma I_{p,q}$
5: $\quad \hat{I}_{p,q} \leftarrow \gamma I_{p,q}$
6: $\hat{I}_{p,q} \leftarrow \hat{I}p, q + I^D$ ▷ Optional

The maximum value in the dirty image, $I^D$, is found and its indices are stored as $p, q$. A scaled interferometer response of the point at $p, q$ (PSF convolved with only that point) is then subtracted from the dirty image. The scaling factor (or CLEAN gain in the literature), $\gamma$, is multiplied to solve the problem iteratively, and therefore to encourage convergence. It is usually set somewhere between 0.1 and 0.5.

This $\gamma$ scaled intensity is then added at point $p, q$ to our candidate image, $\hat{I}$ (or "Model" image in the literature, which is initially empty).

If there is still a point above a user-specified threshold in the dirty image, this process is repeated to clean this point. This continues until the user-specified threshold is met (or a maximal number of iterations is met).

Once all selected points have been added to the candidate image, we convolve it with a "restoring beam" (also called a "CLEAN beam"), which is usually a 2D elliptical Gaussian which best approximates the PSF central lobe.

The remaining residual can then be added onto the candidate image; however, this step is not always performed.

One can observe that the CLEAN algorithm is a Compressed Sensing approach (indeed, it is similar to a matching pursuit algorithm with one atom, the PSF, see Lannes et al. [1997] and Mallat and Zhang [1993]), albeit one which was developed independently of the CS theory.

Clark [1980] developed an optimisation to the Högbom CLEAN algorithm. Since applying a convolution to an image is often more efficient using 2-dimensional FFTs, the idea is to identify

[1] The notation for various terms has changed from the previous sections as CLEAN was developed separately from the CS theory.

a number of components using some faster approximation to the PSF and then convolving them all with the PSF at once (using an FFT).

Clark's algorithm initially selects the smallest patch of the PSF such that it contains the highest exterior sidelobe of the PSF.

As in the Högbom CLEAN, the point with a maximum value is selected.

Instead of subtracting the full-extent PSF from this point (as in Högbom CLEAN) only a beam patch is subtracted (still using a gain factor). Because the beam patch usually has significantly fewer pixels than the PSF, this can be computed much faster.

A new point is selected and reduced in this fashion until the peak value is less than the image's initial peak value scaled by the highest exterior sidelobe of the PSF.

At this point, the algorithm proceeds to a *major cycle* in which the selected point is convolved with the full PSF (using an FFT) and subtracted from the initial residue. Since the dirty image's main peaks will have been reduced, it can continue with more *minor cycles* as above.

Schwab [1984] developed a variant (called the Cotton-Schwab CLEAN) of the Clark algorithm in which the major cycle subtraction is performed on the ungridded visibility samples. This route allows for the correction of gridding errors but introduce a gridding/degridding process at each major cycle, which increases the computation time when compared to performing a single FFT in the gridded images.

Conway et al. [1990] developed a multi-frequency variant of CLEAN which accounts for the artefacts introduced by the intensity variation of sources over a narrow band of frequencies ($\pm 12.5\%$). This allows one to image using the $u - v$ coverage of multiple spectral channels, which is particularly important for sparse arrays or short observation times (which thus have poor $u - v$ coverage).

Cornwell [2008] developed a multi-scale variant of Högbom CLEAN in which interferometer responses at several scales are precomputed and searched for when finding the maximum element. If the response for a larger scale is greater than that of a smaller scale, the larger response is removed from the dirty image instead. This variant can produce superior results when the image contains extended structure.

Rau and Cornwell [2011] combined the ideas used in both the multi-frequency and in the multi-scale variants of CLEAN, producing a multi-scale multi-frequency variant of CLEAN.

### 2.3.2 Maximum Entropy Method

The Maximum Entropy Method (MEM) is a deconvolution technique in which we find the image of maximum *entropy* which fits the data to within the noise level. The term entropy (which should not be confused with physical entropy, though the cost function is derived from the physical definition) is something which, when maximised, introduces legitimate *a priori* information. Narayan and Nityananda [1984] find that one of the best entropy forms for general purpose use is

$$F = -\sum_k I_k \ln \frac{I_k}{M_k e}$$

where $I_k$ is the deconvolved image and $M_k$ is a *default* image which allows a priori information to be incorporated. A low resolution image of the target can provide a good default image.

The resulting intensity of the model should be consistent with the visibility data, which is measured through a $\chi^2$ statistic. This is the measure of the mean-squared difference between the measurements, $V$, and the corresponding values for the model, $V'$:

$$\chi^2 = \sum_k \frac{|V_k - {V'}_k|^2}{{\sigma_k}^2}$$

Obtaining the solution is an iterative procedure such as CLEAN is.

### 2.3.3 Compressed Sensing

Since CS theory has been widely, but unknowingly, used to deconvolve radio astronomy images in the form of CLEAN, one should expect that it has already been shown experimentally to be effective. This is due to the fact that images of the sky are typically sparse, that is, they are mostly point sources placed on a background of low-intensity noise.

Some of these algorithms rely on wavelet representations. They are ultimately not optimal for point sources (which are the sparsest in the image domain) but can be extremely efficient to enforce sparsity on extended emission. When dealing with point sources only, CLEAN is still competitive compared to such CS algorithms.

As such, further development of CS algorithms specifically developed and based on CS theory can produce superior results to CLEAN (which was developed before CS theory was understood). To that end, we adapt the OMP algorithm to deconvolution in Chapter 4.

## 2.4 Literature Review

A large number of deconvolution methods have been developed, often based on particular assumptions about the observed sky. Starck et al. [2002] provided a review of many of the algorithms prior to compressed sensing theory, detailing the formalism, strengths and shortcomings of each of them. They make the point that each method is typically based on a particular noise model, or on particular *a priori* assumptions, under which the particular algorithm will perform best.

They expand many of the algorithms into a multi-scale variant using wavelets. By using a particular wavelet basis, they are able to overcome the major shortcomings of each method.

However, a few years later, Candès [2006] introduced Compressed Sensing theory, a method for reconstructing sparse signals using fewer measurements than is required by the more general Nyquist-Shannon sampling theorem. Since images of the sky are typically sparse, or at least have a sparse representation in some space, this theory can be used for deconvolution.

In addition, Candès [2006] introduced the basis pursuit (BP) algorithm, a method which uses the Compressed Sensing theory to reconstruct a sparse signal.

Tropp et al. [2007] provided a description of OMP, a greedy Compressed Sensing technique for general signal recovery. They also provided detailed experimental results on how many measurements are required to recover a signal.

Wiaux et al. [2009] provided a detailed explanation of compressed sensing from the perspective of radio astronomy, and construct a basis pursuit algorithm. They then used this algorithm to deconvolve both a dirty image containing point sources and a dirty image containing extended structure, and showed that it produces a superior signal-to-noise ratio to CLEAN.

Schwardt [2012] created a CPU implementation of OMP for radio interferometric data. In addition, he benchmarked the performance of OMP against other compressed sensing algorithms ($\text{QP}_\lambda$ and $\text{BP}_\epsilon$), as well as against the Cotton-Schwab CLEAN algorithm. He found that OMP produced sparser images in less time when compared to the other test algorithms, although the image's dynamic range was less than that of the other compressed sensing algorithms.

Schwardt's algorithm executed in 5.3s for a $100 \times 100$ image, executing for 200 iterations using a 2.3 GHz Intel Core i7 CPU. While he did not provide many implementation details, he specified using an FFT, so we can assume it's at least $n \log_2 n$. Extrapolating from this single result implies his implementation (using 200 iterations) would require around an hour for a 4 megapixel image (by contrast, our implementation can manage this in 2.6s) which, while feasible, is still a long time and can become infeasible for larger images.

Schwardt also found that introducing a positivity constraint on the algorithm (thereby making it identical to Non-Negative Least Squares (NNLS) algorithm described by Lawson and Hanson [1974]) allowed convergence with fewer iterations and produced an image with greater dynamic range. This constraint limits the application to non-negative images however, which is not always the case for radio interferometry.

Carrillo et al. [2012] observed that, depending on the observation, astronomical phenomena are typically sparse in either the Dirac basis, wavelet bases, or exhibit gradient sparsity. In particular, astronomical images can contain all these types of structures at once. As such, they define a dictionary of bases in which the image might be sparse, and identify the solution which has the best average sparsity across all chosen bases using a reweighed basis pursuit method. They found that this approach produces superior results to methods only optimising a single basis.

Garsden et al. [2015] tested a CS implementation against Cotton-Schwab CLEAN (CoSch-CLEAN) and Multi-Scale CLEAN (MS-CLEAN). They develop a wavelet based implementation of the FISTA algorithm, a method which replace the CS $\ell_1$ with a smooth approximation, thereby enabling many existing convex-optimisation algorithms to solve the compressed sensing problem.

When deconvolving point sources, they found that their CS implementation performed competitively when compared to CoSch-CLEAN. The CS implementation had an improved angular resolution at high and moderate signal-to-noise ratios, with a similar angular resolution at low ratios. The CS implementation was able to detect more faint sources than CoSch-CLEAN, but had a larger error on their flux density values. When deconvolving extended emissions,

they found the CS implementation showed an improved angular resolution compared to both CoSch-CLEAN and MS-CLEAN.

In order to perform deconvolution faster, hardware solutions become attractive. However, the current implementations are unsuited to deconvolution.

Septimus and Steinberg [2010] implemented OMP on a GPU and on FPGAs; however, this only supports images smaller than 128 elements, with a sparsity no greater than 5, which is too restrictive for radio astronomy as it only allows for images of around $11 \times 11$ pixels and 5 non-zero sources.

Fang et al. [2011] obtained a 30 to 50 times speedup for their GPU implementation of compressed sensing. Their implementation is not specialised for radio interferometric data, and thus does not take into account the algorithmic enhancements described in Chapter 4. As such, it has a worse algorithmic complexity than can be achieved by a specialised method.

Their GPU implementation executed in 1454ms for an "image" of size 16384 (that is, the equivalent of a $128 \times 128$ image), with 4096 measurements and executing for 512 iterations an NVIDIA GTX480 GPU. Extrapolating the results for this general algorithm to a megapixel image would result in an execution time somewhere on the order of months, which is again infeasible for most radio astronomy.

# Chapter 3

# General Purpose Graphics Processing Units

*Graphics Processing Units* (GPUs) have recently received a lot of attention as scientific computing devices. These devices often have thousands of parallel processing units, which allow for rapid computation of parallelisable algorithms when compared to traditional CPUs.

In particular, GPUs are effective when most of the computational time of the algorithm can be reduced to Single Instruction Multiple Data (SIMD) problems, that is, problems in which a single set of instructions needs to be applied to multiple separate data elements.

Since we will be using NVIDIA's GPUs with their *Compute Unified Device Architecture* (CUDA), the details will be for their architecture and conventions, though other GPGPU paradigms are similar.

## 3.1 Applicability

The CS algorithms mostly consist of matrix operations. These operations usually accelerate well when implemented on a GPU [Fatahalian et al., 2004]. This is because such operations typically involve a large number of calculations (relative to memory operations), and typically have a significant amount of inherent parallelism.

As such, an effective GPU algorithm for CS deconvolution should perform much faster than any CPU implementation, which should allow for faster image synthesis.

## 3.2 Compute Architecture

A CPU and a GPU are designed to perform different tasks; as such, they have been designed very differently. A high-level overview of the two is shown in Figure 3.2.1.

A GPU differs from a CPU mainly in the number of *Arithmetic Logic Units* (ALUs), which perform the calculations. A CPU has only a few complex ALUs (typically 2-16), while a GPU has many more simpler cores (NVIDIA's K80, for instance, has almost 5000 cores). It should be noted that CPU cores can be sufficiently complex such that a single core is able to exploit

some parallelism. The AVX instructions on Intel's Ivy Bridge CPUs, for instance, can perform up to 16 multiply-accumulate instruction (32 flops) in a single clock cycle. The theoretical peak performance of both GPUs and CPUs are computed using the number of MAC instructions which can be performed in one second.

The control circuitry is also much more complex on a CPU than on a GPU, allowing the CPU to perform tasks such as branch prediction in order to achieve some parallelism even on sequential code.

The cores of a GPU are collected into *Streaming Multiprocessors* (SMs), each having have independent controllers and caches. The *GigaThread Engine* manages the scheduling of CUDA blocks (see Section 3.3.2) onto the SMs. An overview of NVIDIA's GM204 (Maxwell) architecture is shown in Figure 3.2.2, as well as a detailed view of the *Maxwell SM* (SMM). Table 3.1 list the compute capability specifications for various NVIDIA micro-architectures.

The SMs have a number of components:

- The ALU (Core) computes the standard floating point and integer calculations, including add, multiply, multiply-add, bit-shifting, AND, OR, XOR, etc.
- The *Load-Store units* (LDST) allows values from memory to be read from and written to.
- The *Special Function Units* (SFU) support certain functions commonly used in graphics, including the base-2 logarithm, the base-2 exponential, sine and cosine.
- The *Warp Schedulers* and *Dispatch Units* assign threads (see Section 3.3.1) to the cores.
- The various memory banks (L1 cache, Texture cache, Shared Memory) are used to implement the CUDA memory models described in Section 3.5.
- The *Texture Units* (Tex) are used to perform texture look-ups from the texture memory (see Section 3.5.3).
- The *PolyMorph Engine* is used for graphics rendering, and is not accessible from CUDA.

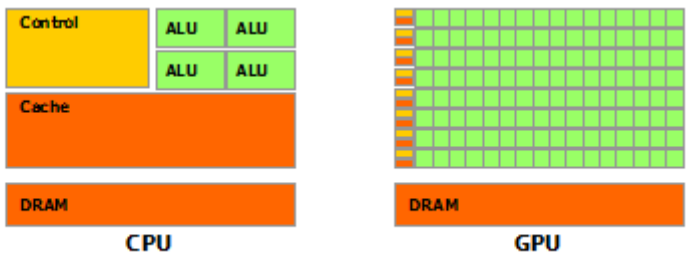



Figure 3.2.1: CPU and GPU architecture. From the CUDA C programming guide [NVI, 2014b].


Figure 3.2.2: GM204 architecture overview (left), with one of the SMMs expanded (right). From the GTX 980 whitepaper [NVI, 2014a].

| | Compute Capability | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Specification** | **1.1** | **1.2** | **1.3** | **2.x** | **3.0** | **3.5** | **5.0** | **5.2** |
| Micro-architecture | Tesla | | | Fermi | Kepler | | Maxwell | |
| Grid Maximum Size | $65535 \times 65535$ | | | $(2^{31} - 1) \times 65535 \times 65535$ | | | | |
| Block Maximum Size | $512 \times 64$ | | | $1024 \times 64$ | | | | |
| Warp Size | 32 | | | | | | | |
| Maximum Blocks per SM | 8 | | | | 16 | | 32 | |
| 32-bit Registers per SM | 8 K | 16 K | | 32 K | 64 K | | | |
| Maximum Shared Mem per SM | 16 KB | | | 48 KB | | | 64 KB | 96 KB |
| Maximum Shared Mem per Block | 16 KB | | | 48 KB | | | | |
| Shared Memory Banks | 16 | | | 32 | | | | |

Table 3.1: CUDA compute capability specifications. Adapted from the CUDA C programming guide [NVI, 2014b].

## 3.3 CUDA

CUDA is programming structure designed to easily make use of NVIDIA's GPUs. It exists as an extension to high-level languages (C, C++, FORTRAN and Python), in order to minimise the additional learning required and to easily integrate into existing code.

In order to construct and understand CUDA applications, it is important to first understand the CUDA work hierarchy.

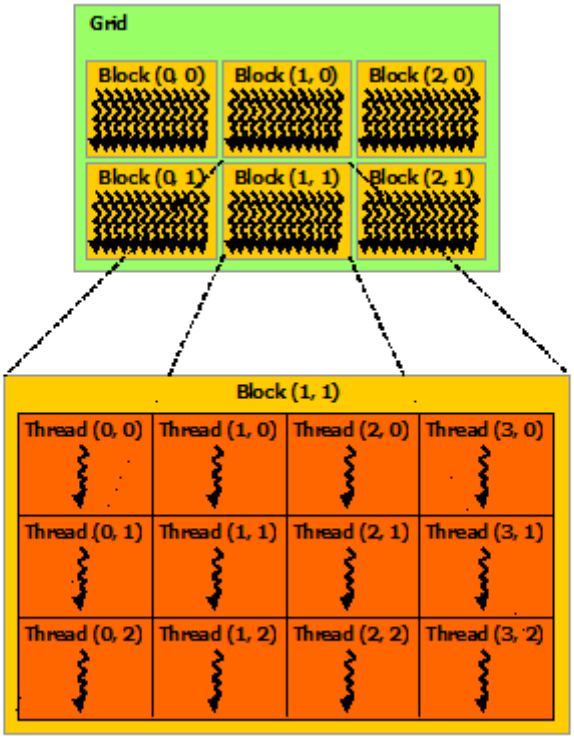



Figure 3.3.1: CUDA work hierarchy. From the CUDA C programming guide [NVI, 2014b]

### 3.3.1 Threads

CUDA organises work into a hierarchy with each basic work item being called a *thread*.

These threads are organised into *warps* (currently containing exactly 32 threads). All the threads in a warp execute the same instruction at the same time. This means that, in the case of conditional logic, *branch divergence* will occur. That is, when some threads execute a different logic path to other threads within the same warp, all threads will have to compute both of the paths.

Threads within a warp are able to read the values in the registers of other threads within the same warp by making use of the Warp Shuffle instructions (only supported by Kepler and Maxwell currently).

### 3.3.2 Blocks

Warps are organised into *blocks*. Blocks contain thread synchronisation constructs, such as shared memory (see Section 3.5 below) and thread barriers, allowing threads within a block to co-operate.

A block is assigned to a single SM (though a single SM can execute multiple blocks simultaneously) and its warps are scheduled onto the available cores by the warp scheduler. Any thread within a block can access the same values in shared memory, or in the L1 cache of the SM on which it is executing, but it cannot access those of other SMs. Since there is no guarantee of which blocks will run on which SMs, separate blocks can only communicate through the relatively slow L2 cache.

### 3.3.3 Kernels

Blocks are organised into a *grid*. Threads in different blocks have almost no communication with each other, and thus should be executing independent sections of work. The execution of a grid is called a *kernel* launch.

Kernels can be launched asynchronously by placing them into different *streams*. This will allow a block from either kernel to be launched on an SM, thereby allowing for a greater utilisation of the GPU if one of the kernels could not fully utilise the GPU by itself.

### 3.3.4 Occupancy

In order to best utilise the GPU, the grid size and block size should be carefully chosen. The idea is to have the maximum number of warps running on each SM simultaneously, while not exceeding any of the SMs resources (such as the shared memory). CUDA 6.5 now provides tools to calculate this automatically.

In some cases this could mean that performing additional calculations in order to use less memory can result in a greater computational throughput, or additionally, using a different, slightly slower memory type can result in more warps running simultaneously.

## 3.4 Memory Access Patterns

Since there are a large number of threads that might access memory at the same time, CUDA provides several memory access patterns to accelerate this access, which algorithms should be designed to take into account.

### 3.4.1 Coalesced Memory Access

*Coalesced memory access* allows neighbouring threads in a warp to access neighbouring memory addresses in a single read/write operation. That is, if thread $i$ accesses memory address $j$, and thread $i + 1$ accesses memory address $j + 1$, these operations can be coalesced into a single operation, up to all of the threads within a single warp. This case is shown in Figure 3.4.1. It should be noted that if the access is not aligned with an integer number of cache lines (represented by blocks in the figure), this access can require an extra memory read.

Figure 3.4.2 shows an example of strided memory access with a stride of 2. In this example, the first half-warp of threads will load an entire cache line, only to use half of the values and, in effect, wasting the bandwidth by reading the other unused values. The second half-warp would then require a second memory operation. This effect is worse with larger strides, as more memory operations would be required. An example of this performance degradation is shown in Figure 3.4.3. Randomly accessing memory elements would produce similar results to a stride the size of the cache line (in words).

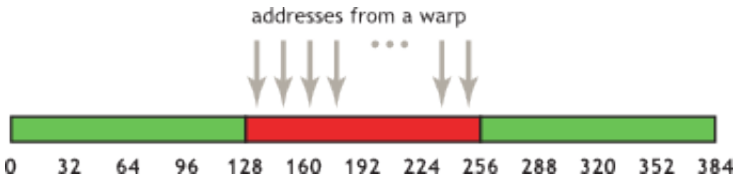



Figure 3.4.1: Example of aligned, coalesced access. From the CUDA C programming guide [NVI, 2014b].

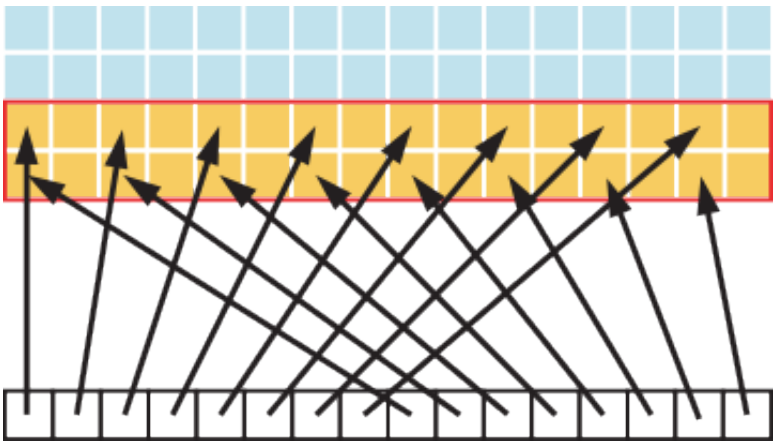


Figure 3.4.2: Example of strided memory access with a stride of 2. From the CUDA C programming guide [NVI, 2014b].

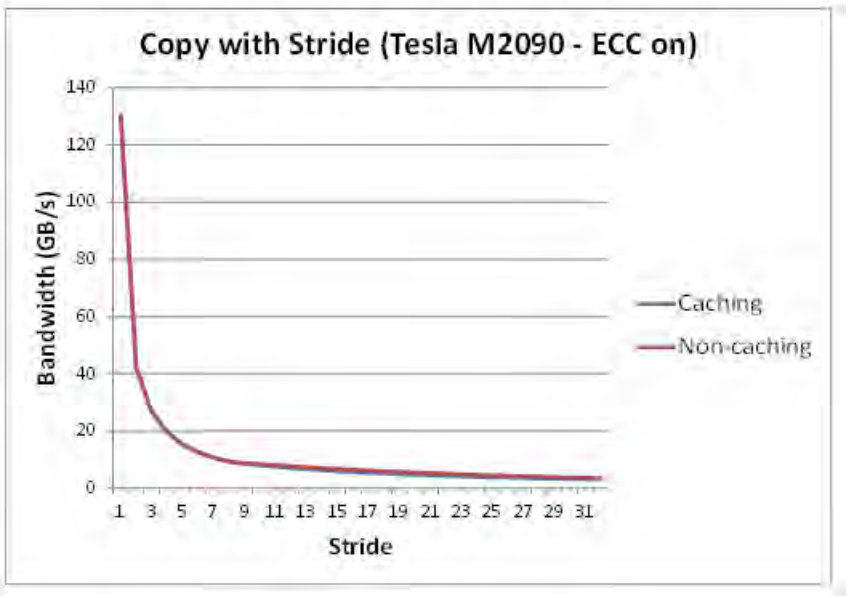



Figure 3.4.3: An example of memory bandwidth performance for strided memory access for various strides. Caching the blocks containing each accessed thread provides no performance benefit in this case. From the CUDA C programming guide [NVI, 2014b].

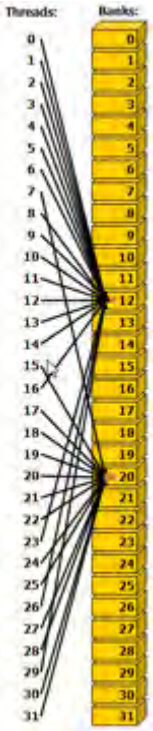



Figure 3.4.4: An example of multiple threads accessing a single memory bank. If the threads access the same address, this results in a broadcast. If the threads access a different memory address, this results in a bank conflict. From the CUDA C programming guide [NVI, 2014b].

### 3.4.2 Memory Banks

*Memory banks* are separate sections of memory which can be read from simultaneously. When two threads within a block attempt to access different memory addresses within the same bank of memory (as is shown in Figure 3.4.4), it results in what is called a *bank conflict*, and two operations are needed.

If, however, the threads attempt to access the same memory address, the bank can *broadcast* the memory value to allow multiple threads within a warp to all read the value in a single read operation.

## 3.5 Memory Types

CUDA has a number of different memory models which are effective in different scenarios. Figure 3.5.1 provides a diagrammatic overview of the various models.

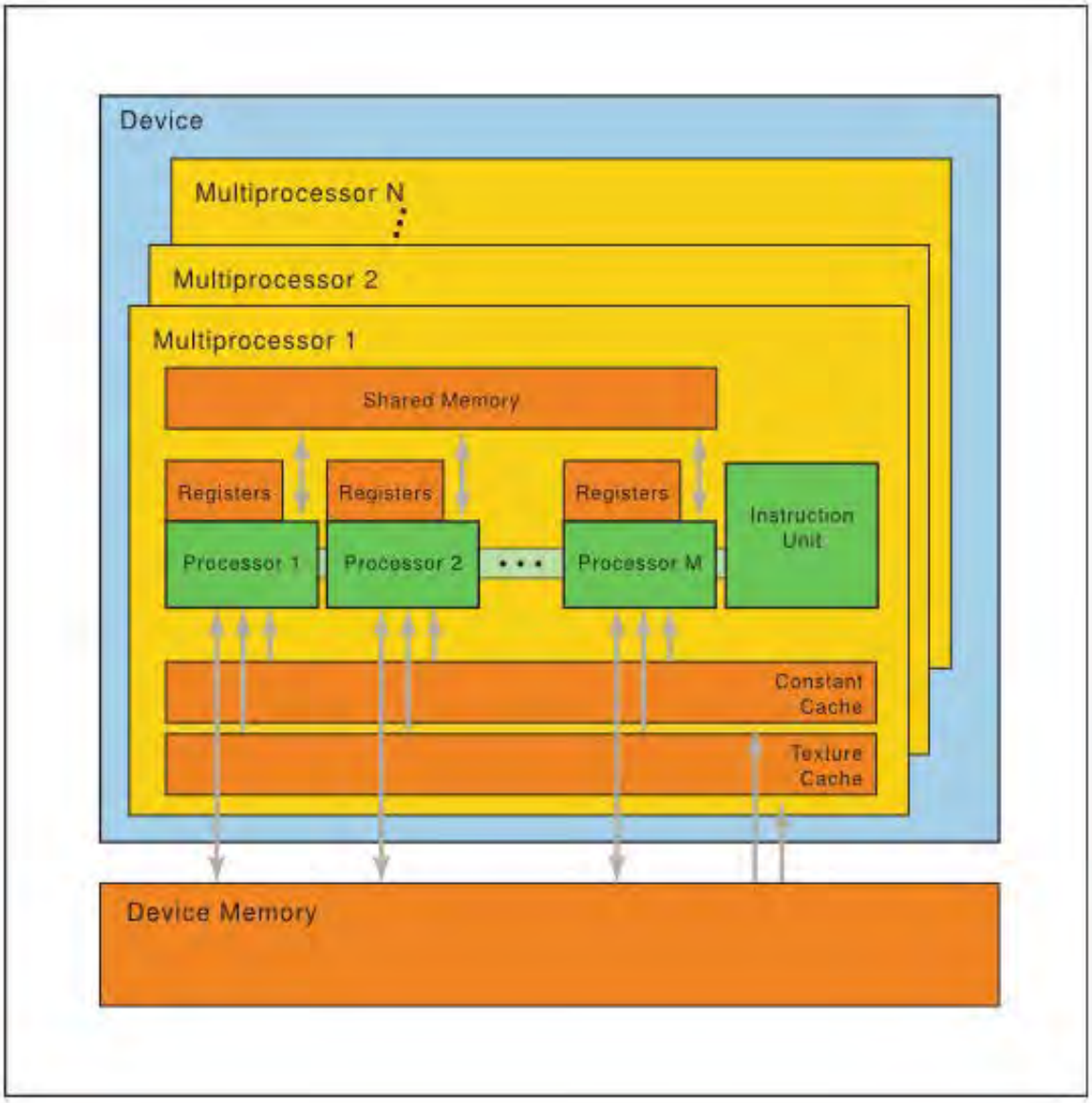



Figure 3.5.1: CUDA memory model. From the CUDA Parallel Thread Execution ISA [NVI, 2014c].

### 3.5.1 Global/Device Memory

*Global Memory* is the largest and slowest store of GPU memory, usually consisting of a few gigabytes. It has two levels of cache: a larger (a few megabytes), slower L2 cache, and a smaller, faster L1 cache. Global memory supports coalesced access and (through its cache) broadcasting.

Memory can be transferred between the CPU's (host) memory into GPU memory. In addition, an asynchronous transfer will allow the GPU to perform work while the memory is being transferred.

This memory is typically used to store the main input and output of the kernel but, if possible, it is usually better to store intermediate results elsewhere.

### 3.5.2 Shared Memory

*Shared Memory* is a small store of memory (currently 0-96 KB) which is shared between threads within a block. Maxwell GPUs have a dedicated shared memory, while Fermi and Kepler GPUs, shared memory splits its memory with the L1 cache, and the ratio between them can be set. Shared memory has several (currently 32) memory banks, with sequential memory addresses falling into sequential banks; is also supports broadcasting.

Shared memory is typically used to store values which will be used later by another thread (in the same block), or to store intermediate results when separate threads are co-operating in a calculation.

### 3.5.3 Texture Memory

*Texture Memory* is a memory store designed to store continuous 1D, 2D or 3D data. It is stored in global memory, but cached in the texture cache (which is shared with the L1 cache on Maxwell architectures). It is cached in 1D, 2D, or 3D blocks (depending on the texture mode).

Its indices can be normalised to be in a $[0, 1)$ range, and support linear interpolation between the nearest values in the same time as a normal memory read. This interpolation is between the nearest 2, 4 or 8 points for 1D, 2D or 3D data respectively.

Texture memory, which is considered read-only while a kernel is running, can only be modified outside of a kernel launch.

Texture cache does suffer some additional latency compared to shared memory and the L1 cache.

### 3.5.4 Constant Memory

*Constant Memory* is a small store of read-only memory. It is stored in global memory and usually cached in the texture cache. Due to its read-only nature, it can be accessed by any thread, and is cached identically across all of the multiprocessors.

Although it is usually stored in the texture cache, it does not support multidimensionality, index normalisation or linear interpolation, and thus does not suffer the same latency penalty. Its latency is thus similar to that of shared memory and the L1 cache.

### 3.5.5 Registers

*Registers* each contain a single memory value. Each thread has independent registers, although they can swap registers (as described in Section 3.3.1. If more registers are allocated than there is memory available, registers can spill into the L1 cache, the L2 cache, and global memory.

Registers have no latency (unless a spilled register needs to be retrieved).

# Chapter 4

# Adapting OMP to Radio Interferometry

This chapter will detail the adaptation of the Orthogonal Matching Pursuit to deconvolution of radio interferometric images. In particular, we are adapting Algorithm 2.2.2 to this use case. In terms of compressed sensing, the measurement system ($\Psi$) is a "masked" 2-dimensional Fourier transform matrix, the sensing matrix ($\Phi$) is the identity matrix, and $y$ is a vector containing the visibilities produced by the interferometer.

This particular choice allows us to make some improvements to OMP. In this chapter we detail each step shown in Figure 4.0.1, a diagrammatic overview of OMP.

We initially project the difference by which each candidate component can reduce the residuals. This component is then added to the list of selected pixels. From there, we determine the weights for each selected component that would minimise the residue. This continues until the stopping case is reached. This stopping case is typically a fixed number of iterations.

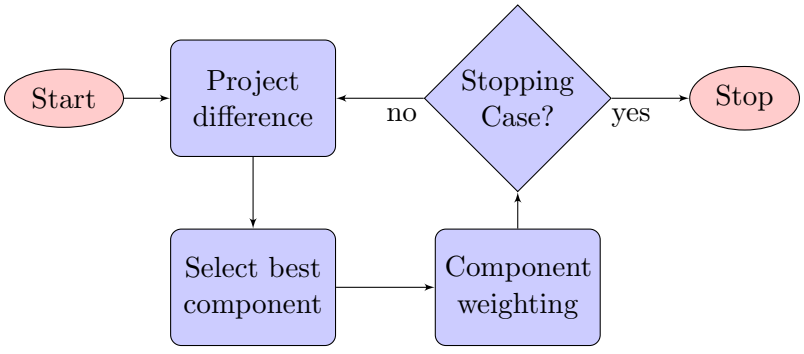



Figure 4.0.1: A diagrammatic overview of OMP.

## 4.1 Definitions

Let $y$ be a vector, of size $m$, which represents the measurements, and let the image we want to reconstruct be of size $n$ (that is, a $\sqrt{n} \times \sqrt{n}$ image). Let $F$ be the 2-dimensional $n \times n$ Fourier transform matrix (which is multiplied by the size $n$ image vector) and $\mathbf{M}$ the $m \times n$ masking

matrix that picks the $m$ measured visibilities from the Fourier plane. Note that $F^{\mathrm{H}}$ is then the inverse Fourier transform. As such, we have $\Psi = \mathbf{MF}$.

For iteration $i$, let $x_i$ represent our candidate image, which is initially blank (every entry is zero), and let $r_i$ represent the residual image of the candidate $x_i$ (and thus is initially the dirty image). That is, $r_i$ is the difference between the measurements ($y$) and the measurements that would be obtained from the candidate image ($\mathbf{MF}x_i$), thus $r_i = y - \mathbf{MF}x_i$. In the final formulation, we do not directly need $r_i$ and instead use its Fourier transform. This allows us to avoid the computational cost of de-gridding and re-gridding the image.

Let $P$ be the matrix representing the 2-dimensional convolution by the Point Spread Function (PSF).

Artefacts are considered to be any features in the dirty image that do not correspond to a convolution of an image by the PSF. These can be caused by noise or as a result of using an approximation in the gridding process.

## 4.2 Projection

The first step of OMP is to identify which column of $\mathbf{MF}$ will most reduce $r_i$. For column $c_i$, the projection onto $r_i$ can be computed (according to the general OMP approach) as:

$$\frac{c_i \cdot r_i}{\|c_i\|_2}.$$

However, in interferometry we have the property that each entry of $c_i$ is an entry from $F$ and thus $\|c_i\|_2 = \frac{m}{\sqrt{n}}$, a constant factor for any $n$ and can thus be ignored when identifying the maximum element. Thus we can compute the weighting of each column as the vector:

$$(\mathbf{MF})^{\mathrm{H}} r_i = \mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} r_i.$$

Since we have $r_i = y - \mathbf{MF}x_i$, we get:

$$\begin{aligned} \mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} r_i. &= \mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} (y - \mathbf{MF}x_i) \\ &= \mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} y - \mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}}\mathbf{MF}x_i \end{aligned}$$

where $\mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} y$ is the dirty image, and $\mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}}\mathbf{MF}x_i$ is $x_i$ convolved with the PSF.

The best column can then be selected by finding the index, $i$, of the maximum value in $\mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}} r_i$. We can then construct $\mathbf{A}_i$ as the matrix that selects values whose indices correspond to those selected in both the current and in previous iterations (satisfying $D$ in Algorithm 2.2.2).

Due to the implementation of gridding and the nature of the radio antennas, artefacts may be produced around the edges of the dirty image (which can be seen in figure 6.1.2). This can be mitigated by restricting the selection to a window inside the projection, that is, to not search for the maximum near the boundaries of the image.

## 4.3 Weighting

We now need to identify the weights for the selected columns, $x_{i+1}$, which best approximate the measurements. We identify this by finding the parameters which minimise the $\ell_2$ norm:

$$\arg\min_{x_{i+1}} \|y - \mathbf{MFA}_i x_{i+1}\|_2$$

These can be computed by solving the normal equation:

$$(\mathbf{MFA}_i)^{\mathrm{H}} \mathbf{MFA}_i x_{i+1} = (\mathbf{MFA}_i)^{\mathrm{H}} y.$$

General OMP would have us calculate this product directly, however, because of the specific matrices used in interferometry, we can simplify this further. By expanding we get:

$$\mathbf{A}_i^{\mathrm{H}} \mathbf{F}^{\mathrm{H}} \mathbf{M}^{\mathrm{H}} \mathbf{MFA}_i x_{i+1} = \mathbf{A}_i^{\mathrm{H}} \mathbf{F}^{\mathrm{H}} \mathbf{M}^{\mathrm{H}} y. \tag{4.1}$$

Since $\mathbf{M}^{\mathrm{H}}\mathbf{M}$ is a diagonal matrix, it can be considered as a point-wise multiplication by the vector, $s$, which contains the elements along the diagonal of $\mathbf{M}^{\mathrm{H}}\mathbf{M}$ (in particular, point-wise multiplication by $s$ corresponds to applying the sampling function of the observation). By the convolution theorem, this is identical to an inverse Fourier transform, a convolution by the kernel $\mathbf{F}s$ (which is the PSF), followed by a Fourier transform. That is:

$$\mathbf{M}^{\mathrm{H}}\mathbf{M} = \mathbf{FPF}^{\mathrm{H}}$$

where $\mathbf{P}$ represents the 2D convolution by the Fourier transform of $s$.

Thus Equation 4.1 becomes:

$$\mathbf{A}_i^{\mathrm{H}} \mathbf{F}^{\mathrm{H}} \mathbf{FPF}^{\mathrm{H}} \mathbf{FA}_i x_{i+1} = \mathbf{A}_i^{\mathrm{H}} \mathbf{F}^{\mathrm{H}} \mathbf{M}^{\mathrm{H}} y.$$

Since $F$ and $F^{\mathrm{H}}$ are inverse to each other, this simplifies to:

$$\mathbf{A}_i^{\mathrm{H}} \mathbf{PA}_i x_{i+1} = \mathbf{A}_i^{\mathrm{H}} \mathbf{F}^{\mathrm{H}} \mathbf{M}^{\mathrm{H}} y. \tag{4.2}$$

Since $\mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}}$ remains fixed between iterations, it can be precomputed and stored. Also, since $\mathbf{P}$ is a 2D-convolution, only the kernel needs to be stored instead of the full matrix.

Furthermore, since $\mathbf{A}_i$ only has a single non-zero value for any column, $\mathbf{A}_i^{\mathrm{H}}\mathbf{PA}_i$ consists simply of scaled values from $\mathbf{P}$ and can be constructed directly (rather than by matrix multiplication).

Between two iterations, exactly one row and one column is added to $\mathbf{A}_i^{\mathrm{H}}\mathbf{PA}_i$. If we solve the system of equations using a matrix inverse, we can thus store the inverse and simply update it on each iteration (the details doing so are shown in Section 5.1).

## 4.4 Stopping Case

There are several possible stopping cases, depending on what the astronomer requires. The best stopping case, of course, is when the residue is blank and the problem has been perfectly solved. This cannot occur in practice, since there is always inherent noise in the measurements. As such, it is sometimes better to only run for a specific number of iterations, which can be manually tweaked if a first pass produces a bad image. Other options are to stop when further iterations no longer produce much change to the error, or when the error is sufficiently small.

## 4.5 Complexity

For iteration $i$ we have:

1. The projection step requires a convolution operation, which can be performed using zero-padded FFTs $[O(n \log n)]$, followed by a difference calculation $[O(n)]$, followed by finding the maximum $[O(n)]$. Thus this step is $O(n \log n)$.

2. The weighting step requires selecting the values to be added to the matrix $[O(i)]$, followed by updating the Matrix inverse $[\mathrm{O}(i^2)]$, and then matrix-vector multiplication $[O(i^2)]$. Thus this step is $O(i^2)$.

3. Determining if the stopping case is reached might require computation of the $\ell_2$ norm $[O(n)]$.

Thus, when executing for $k$ iterations, the entire deconvolution has a complexity of:

$$\begin{aligned} & O\left(\sum_{i=1}^{k}\left[n \log n + i^2\right]\right) \\ = \; & O\left(kn \log n + \frac{k(k+1)(2k+1)}{6}\right) \\ = \; & O\left(kn \log n + k^3\right) \end{aligned}$$

For the most part, the complexity is dominated by the $kn \log n$ part determined by a linear number of Fourier transforms. However, in the case of a large number of iterations (typically many thousands for standard image sizes), the $k^3$ part may start to dominate, making each successive iteration significantly more costly than the previous one.

# Chapter 5

# Implementation

In this chapter, we discuss the particulars of the implementation. We review the least-squares implementation used in computing the optimal weights for the selected pixels. We discuss the final method (matrix inversion) used in the implementation as well as the Cholesky decomposition which was implemented but discarded due to lack of parallelisability.

We also discuss the methods and libraries used in the implementation, with a focus on how the parallelisation was performed for both CPU and GPU architectures. We also look at the expected memory utilisation for various image sizes and iteration counts.

## 5.1 Least Squares Implementation

In order to obtain the best weights for the selected columns, we need to solve Equation 4.2. Normally we would use a matrix inversion or a Cholesky decomposition in order to solve this, which is a $O(i^3)$ computation. This kind of calculation becomes prohibitively expensive for large matrices.

Fortunately, the matrix we need to invert, $\mathbf{R}_i = {\mathbf{A}_i}^{\mathrm{H}}\mathbf{P}\mathbf{A}_i$, is similar to the matrix already inverted in the previous iteration, $\mathbf{R}_{i-1}$. In particular, since $\mathbf{R}_i$ is constructed as $\mathbf{R}_i = \mathbf{H}_i^{\mathrm{H}}\mathbf{H}_i$ (for $\mathbf{H}_i = \mathbf{M}\mathbf{F}\mathbf{A}_i$) in which $\mathbf{H}_i$ has had a single column, $f_i$, added to it, we are able to update the inverse obtained in the previous step by using the method described by Fang et al. [2011]:

$$ {\mathbf{R}_i}^{-1} = \left( \begin{array}{c|c} \mathbf{F} & -d{\mathbf{R}_{i-1}}^{-1}{\mathbf{H}_{i-1}}^{\mathrm{H}}c_i \\ \hline -dc_i{}^{\mathrm{H}}\mathbf{H}_{i-1}{\mathbf{R}_{i-1}}^{-1^{\mathrm{H}}} & d \end{array} \right) $$

where

$$ d = \frac{1}{{\|c_i\|_2}^2 - c_i{}^{\mathrm{H}}\mathbf{H}_{i-1}{\mathbf{R}_{i-1}}^{-1}{\mathbf{H}_{i-1}}^{\mathrm{H}}c_i} $$

$$ \mathbf{F} = {\mathbf{R}_{i-1}}^{\mathrm{H}} + d{\mathbf{R}_{i-1}}^{-1}{\mathbf{H}_{i-1}}^{\mathrm{H}}c_i c_i{}^{\mathrm{H}}\mathbf{H}_{i-1}{\mathbf{R}_{i-1}}^{-1^{\mathrm{H}}} $$

There are a number of repeated calculations in this formulation. To consolidate terms, we let:

$$ u_1 = {\mathbf{H}_{i-1}}^{\mathrm{H}}c_i $$

We do not need to explicitly store $\mathbf{H}_{i-1}$ or $c_i$, as we can obtain this product directly. Since $c_i$ is a column of $\mathbf{MF}$, $u_1$ is the corresponding row in:

$$\begin{aligned}\mathbf{H}_{i-1}{}^{\mathrm{H}}\mathbf{MF} &= (\mathbf{MFA}_i)^{\mathrm{H}}\mathbf{MF} \\ &= \mathbf{A}_i{}^{\mathrm{H}}\mathbf{F}^{\mathrm{H}}\mathbf{M}^{\mathrm{H}}\mathbf{MF} \\ &= \mathbf{A}_i{}^{\mathrm{H}}\mathbf{P}\end{aligned}$$

and the values can thus be obtained directly from the PSF. We then let:

$$u_2 = \mathbf{R}_{i-1}{}^{-1}\mathbf{H}_{i-1}{}^{\mathrm{H}}c_i = \mathbf{R}_{i-1}{}^{-1}u_1.$$

We also note that, when the PSF is considered as the Fourier transform of the Sampling Function, $\|c_i\|_2{}^2$ is simply the zero-frequency component (centre) of the PSF.

By substituting $u_1$ and $u_2$ into the initial formulation, we then get:

$$\mathbf{R}_i{}^{-1} = \left(\begin{array}{c|c} \mathbf{F} & -du_2 \\ \hline -du_2{}^{\mathrm{H}} & d \end{array}\right)$$

where

$$d = \frac{1}{\|c_i\|_2{}^2 - u_1{}^{\mathrm{H}}u_2}$$

$$\mathbf{F} = \mathbf{R}_{i-1}{}^{\mathrm{H}} + du_2u_2{}^{\mathrm{H}}$$

### 5.1.1 Updating the Cholesky decomposition

A Cholesky decomposition can also be updated in a similar fashion. This section describes the process. Unfortunately, this formulation does not parallelise as well, and is thus unsuited to multi-core CPUs and GPUs. The method is nevertheless included here for completeness.

Let A be a normal $n \times n$ matrix partitioned as:

$$\mathbf{A} = \left(\begin{array}{c|c} \alpha_{11} & a_{21}{}^{\mathrm{H}} \\ \hline a_{21} & \mathbf{A}_{22} \end{array}\right)$$

such that $\alpha_{11}$ is a scalar, $a_{21}$ is a $n-1$ vector and $\mathbf{A}_{22}$ is a $(n-1) \times (n-1)$ matrix.

As shown in Baker et al. [1998], we can iteratively compute the Cholesky decomposition $\mathbf{LL}^{\mathrm{H}} = \mathbf{A}$ as:

$$\mathbf{L} = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)$$

where

$$\begin{aligned}\lambda_{11} &= \sqrt{\alpha_{11}} \\ l_{21} &= \frac{a_{21}}{\lambda_{11}}\end{aligned}$$

and $\mathbf{L}_{22}$ is the Cholesky decomposition of $\mathbf{A}_{22} - l_{21}{l_{21}}^{\mathrm{H}}$.

Thus, if we write $\mathbf{A}$ and $\mathbf{L}$ as:

$$\mathbf{A} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{pmatrix} \qquad \mathbf{L} = \begin{pmatrix} \lambda_{1,1} & 0 & \cdots & 0 \\ \lambda_{2,1} & \lambda_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n,1} & \lambda_{n,2} & \cdots & \lambda_{n,n} \end{pmatrix}$$

We can compute some $\lambda_{i,j}$ $(i > j)$ by subtracting products of elements from $\mathbf{L}$ corresponding to $i$ and $j$, and then divide by $\lambda_{j,j}$. That is,

$$\lambda_{i,j} = \frac{\alpha_{i,j} - \sum_{k=1}^{j-1} \lambda_{i,k}\overline{\lambda_{j,k}}}{\lambda_{j,j}}, \tag{5.1}$$

where $\overline{\lambda_{j,k}}$ refers to the complex conjugate of $\lambda_{j,k}$. The entries along the diagonal are similar:

$$\lambda_{i,i} = \sqrt{\alpha_{i,j} - \sum_{k=1}^{i-1} \lambda_{i,k}\overline{\lambda_{i,k}}}. \tag{5.2}$$

Since this formulation of $\lambda_{i,j}$ relies only on elements of $\mathbf{L}$ that appear above and to the left (and this will in turn also hold for those entries), it will not change if entries elsewhere in the matrix are changed, or even if the matrix changes size.

Let $\mathbf{A}'$ be the matrix $\mathbf{A}$ which has had one row and one column added, i.e.

$$\mathbf{A}' = \left(\begin{array}{c|c} \mathbf{A} & {a'}^{\mathrm{H}} \\ \hline a' & \alpha' \end{array}\right)$$

where $a'$ is a $n$-vector and $\alpha'$ is a scalar. Having already calculated the Cholesky decomposition $\mathbf{L}\mathbf{L}^{\mathrm{H}} = \mathbf{A}$, we now want to calculate the Cholesky decomposition $\mathbf{L}'{\mathbf{L}'}^{\mathrm{H}} = A'$.

Using Equations 5.1 and 5.2 we derive:

$$\mathbf{L}' = \left(\begin{array}{c|c} \mathbf{L} & 0 \\ \hline l' & \lambda' \end{array}\right) \tag{5.3}$$

where $l'$ is found using Equation 5.1 and $\lambda'$ is found using Equation 5.2.

To parallelize this, we need to compute $l'$ in parallel. However each element in $l'$ requires the elements to the left to have already been calculated. As such, each element must be calculated sequentially, which limits the parallelisibility.

## 5.2 Implementing OMP on the CPU

Since the FFT dominates the computational complexity, this should be the most rigorously optimised step. Fortunately, a lot of effort has already been expended on optimising FFTs, and we can thus use the popular FFTW library [Frigo and Johnson, 1998]. In particular, this library makes good use of multi-core architectures, as well as the AVX instructions found in most modern desktop CPUs, providing similar performance to vendor-tuned code such as Intel's MKL FFT library [Wang et al., 2014].

Furthermore, since the FFTs are being used to compute a convolution, we need to zero-pad the image to twice its initial height and width in order to compute the discrete convolution (since the convolution theorem applies to the circular convolution). Additionally, because we need to know the contribution that a source on one corner of the image will have in the opposite corner of the dirty image, the convolution kernel will have to extend out to twice the dimension of the dirty image; that is, the PSF will have to extend over twice the horizontal and vertical range of the dirty image. This can then be circularly convolved with the zero-padded image in order to obtain the correct discrete convolution. An diagrammatic explanation for this is shown in Figure 5.2.1. The code for this is shown in Listing 5.1. This results in the FFT operating on an image four times the size of the dirty image; this has a significant memory and performance cost.

Although the FFT determines the computational complexity, FFTs can be computed very quickly for reasonably sized images (around 50 million floating point operations for a 1 MP image). This means that the various linear steps (finding the maximum element, computing the $\ell_2$ norm) become significant. To compensate, we compute these steps in parallel using the OpenMP framework [Dagum and Menon, 1998].

OpenMP allows us to easily parallelize the linear for loops, so long as each loop iteration is independent from every other iteration. An example of this is shown in Listing 5.2. In addition, implementing supported reductions is also easily accomplished, an example of which is shown in Listing 5.3. Performing a reduction which is not explicitly catered is somewhat more involved. Listing 5.4 shows the code used to find the index of the maximum element in an array. For this reduction, each OpenMP thread must accumulate into a separate variable (each of which should be in a separate cache line in order to avoid false sharing [Torrellas et al., 1994]) which are then combined.

A clean window can be chosen, which prevents the selection of an element outside this area in the *projection* step of OMP. This can be used if the sources are known to lie within a particular area, or to prevent selection in the border areas of the image in the case of gridding artefacts.

Since the image resolution is typically higher than the resolving power of the interferometer, the model image is often convolved with a Gaussian beam, called the *restoring beam*, to filter out the higher frequency components. Since CS algorithms can bring super-resolution (the algorithm is able to resolve sources within the angular resolution of the interferometer), this convolution is currently raising some discussion and, as such, this choice is often more an aesthetic choice rather than a physical choice.

When computing this convolution using the convolution theorem, the Fourier transform of the Gaussian kernel (that is, the restoring beam shape) can be computed in $uv$ co-ordinates directly (instead of computing the Gaussian kernel in image space, and then computing the FFT of that kernel), since the Fourier transform of a Gaussian is another Gaussian with an inverted standard deviation. This prevents an unnecessary Fourier transform, and doesn't require any extra memory.

The final implementation requires 19 times the memory required to store the image. This excludes the memory required for initial storage of the image and PSF (an additional 5 times the storage of the image), the calculation of the inverse matrix ($2k^2 + 2k$ floating point numbers), and a small constant factor. This means that an image requires 96 MB of memory per megapixel for a 32-bit floating point system. While this is more than sufficient for most current interferometers, which produce images smaller than 16 megapixels, future interferometers are expected to produce far larger images. The Square Kilometre Array (expected to be completed by 2024), for instance, is expected to produce up to 10-gigapixel images. Table 5.1 shows expected memory costs for various image sizes and iteration counts. Table 5.2 shows the breakdown of the memory cost.

```
// Copy image to a buffer of twice the height and width.
memset(img_padded, 0, height*width*4*sizeof(*img_pad));
for (int j=0; j<height; ++j)
  for (int k=0; k<width; ++k)
    img_padded[j*width*2 + k] = img[j*width + k];

// Transform to UV co-ordinates via FFT
fftp.fft2(img_padded, uv_padded);

// Point-multiply by a precomputed FFT of the PSF.
// This precomputed mask is correctly shifted and include a normalisation factor.
for (int j=0; j<height*2*(width+1); ++j)
  uv_padded[j] *= mask_padded[j];

// Transform back to image co-ordinates via FFT
fftp.ifft2(uv_padded, img_padded);

// Copy image back into original buffer
for (int j=0; j<height; ++j)
  for (int k=0; k<width; ++k)
    img[j*width + k] = img_padded[j*width*2 + k];
```

Listing 5.1: Convolution performed using FFT to eliminate the cyclical nature of the FFT. A custom FFTW wrapper is used to compute the FFTs.

```
#pragma omp parallel for
for (int i=0; i<width*height; ++i)
  result[i] = lhs[i] - rhs[i];
```

Listing 5.2: OpenMP code to parellelize a for loop in which each loop iteration is independant from every other iteration (used here to compute a difference image).

```
#pragma omp parallel for reduction(+:sum)
for (int i=0; i<width*height; ++i)
  sum = sum + pow(std::abs(img[i]), 2);
```

Listing 5.3: OpenMP code for reduction into a sum (used here to calculate the $\ell_2$ norm).

```
// Create separate reduction variable (accumulator) for each thread.
//    These variables are separated in memory by the cache line size
//    to prevent false sharing.
int * ix = new int[num_threads * CACHE_LINE_SIZE];
for (int j=0; j<num_threads; ++j)
  ix[j * CACHE_LINE_SIZE] = w_starty*width + w_startx;

// Allow each thread to reduce into a separate accumulator.
#pragma omp parallel for
for (int j=w_starty; j<w_endy; ++j)
  for (int k=w_startx; k<w_endx; ++k)
    if (img[j*width + k] > img[ix[omp_get_thread_num() * CACHE_LINE_SIZE]])
      ix[omp_get_thread_num() * CACHE_LINE_SIZE] = j*width + k;

// Reduce the accumulators to find the index of the maximum element.
int ix_reduce = ix[0];
for (int j=0; j<threads; ++j)
  if (img[ix[j * CACHE_LINE_SIZE]] > img[ix_reduce])
    ix_reduce = ix[j * CACHE_LINE_SIZE];
delete [] ix;
return ix_reduce;
```

Listing 5.4: OpenMP code for a general reduction (used here to find the index of the maximim element). This is somewhat simpler in later gcc compiler versions, which allow custom OpenMP reductions to be specified.

## 5.3 Implementing OMP on the GPU

The GPU implementation was designed by using the CPU implementation as a template. In particular, the unified memory model introduced in CUDA 6 [Harris, 2013] supports incremental

| **Image Size** | 1 iteration | 1000 iterations | 10000 iterations |
|---|---|---|---|
| 1 MP | 96 MB | 104 MB | 896 MB |
| 10 MP | 960 MB | 968 MB | 1.8 GB |
| 100 MP | 9.6 GB | 9.6 GB | 10 GB |
| 1 GP | 98 GB | 98 GB | 99 GB |
| 10 GP | 983 GB | 983 GB | 984 GB |

Table 5.1: Expected memory cost for CPU implementation for various image sizes and algorithm iteration counts.

| **Buffer Use** | **Size** |
|---|---|
| Dirty Image | height × width |
| Deconvolved Image | height × width |
| Residue | height × width |
| PSF | 4 × height × width |
| Padded Image | 4 × height × width |
| FFT of Image | 4 × height × width |
| FFT of PSF | 4 × height × width |
| Inverse Matrix | $k^2$ |
| Matrix Update | $k^2$ |
| Solution Vectors | $2k$ |

Table 5.2: Memory cost breakdown for an image of size $n$ and deconvolving for $k$ iterations. Constant factors are not included.

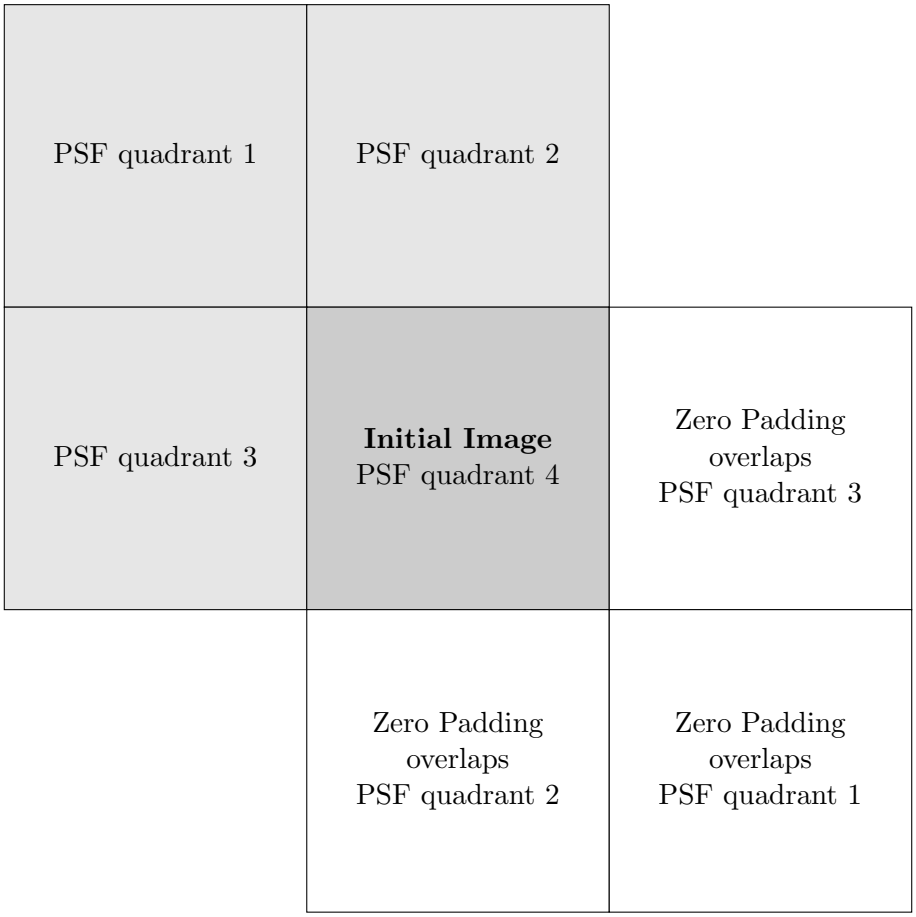



Figure 5.2.1: Example of the PSF convolution applied to the first (top-left) pixel in the image. The PSF needs to cover twice the field of the image, so that any contributions from bottom-left pixel are still included. Furthermore, since a convolution using FFT is a cyclical convolution, the remaining PSF quadrants will overflow to the opposite size of the image. As such, the image should be zero-padded so that this effect can be ignored (any contribution from overflow will be in the zero-padded region, and thus zero).

development by allowing each function to be rewritten for the GPU while still compiling into a functioning program.

CUFFT [Nvidia, 2010] was was chosen to compute the FFTs, both for its performance and for its similarity to FFTW. In particular, it has a similar API to FFTW, thereby allowing for easier implementation from the CPU code; it also uses the same input and output format, preventing additional complexities in the code.

The Thrust [Hoberock and Bell, 2009] library was chosen to compute most of the simple computations. This library (which is based on C++'s Standard Template Library) should allow for an optimised implementation for any CUDA device. In particular, Thrust's reduce function is used to select the column in the *projection* step. Since this selection is restricted to a specified window, a custom iterator was implemented which passes over regions outside this window. Thrust was also used to perform simpler operations such as padding the image for the convolution, performing the point-wise multiplication for the convolution, scaling arrays, setting values of array elements, and for permuting arrays.

Similarly, the matrix-vector multiplication has been implemented using CUDA's CUBLAS [Nvidia, 2008] library. One problem with CUBLAS is that it requires a column-major order, while CUFFT requires a row-major order. Fortunately, all the matrices that CUBLAS operates on are symmetric, and thus the order can safely be ignored.

The calculation of the inverse matrix is a bit more complicated:

1. We need to obtain the $u_1$ from Section 5.1. This is done by simply reading off the appropriate values from the PSF.

2. We can then use $u_1$ to obtain $u_2$ by multiplying $u_1$ by the inverse matrix obtained in the previous iteration, which is achieved using CUBLAS.

3. Using Thrust, we can then take the inner product of $u_1$ and $u_2$ in order to calculate $d$.

4. The previous iteration's inverse matrix must then receive a rank-1 update. Since the result of this update will be the new value in the current iteration, the result is stored directly in a new array allocated for the current iteration's inverse matrix, instead of occurring in-place.

5. We can then fill in the missing row and column of the new inverse matrix using $d$ and $u_2$, completing the calculation of the inverse matrix. Since the previous iteration's inverse matrix is no longer needed, that memory can be cleared.

6. The final step is then to multiply the right-hand side values obtained from the dirty image by the new inverse matrix. This is achieved using CUBLAS.

For all of these kernels, the block size was determined by CUDA in order to maximise the occupancy. This allows the same code to work on multiple architectures without having to optimise for each one, including future, unknown architectures.

| **Image Size** | **Iterations** | | |
|---|---|---|---|
| | 1 | 1000 | 10000 |
| 1 MP | 80 MB | 88 MB | 880 MB |
| 10 MP | 764 MB | 772 MB | 1.6 GB |
| 100 MP | 7.6 GB | 7.6 GB | 8.4 GB |
| 1 GP | 78 GB | 78 GB | 79 GB |
| 10 GP | 778 GB | 778 GB | 779 GB |

Table 5.3: Expected GPU global memory cost for GPU implementation for various image sizes and iteration counts.

Since the final GPU implementation requires the same memory buffers as the CPU implementation, it requires 19 times the GPU memory required to store the image. In addition, it also requires a small logarithmic factor to perform reductions. This excludes the host (CPU) memory required for the initial storage and final result (7 times the storage of the image), the GPU memory required to store the inverse matrix ($2k^2 + 3k$ floating point numbers), and a constant factor. This means that an image requires about 76 MB of GPU memory per megapixel for a 32-bit floating point system. Table 5.3 shows expected GPU memory costs for various image sizes and iteration counts.

Additionally, the execution of the GPU kernels is in a separate stream for each image, allowing for multiple images to be deconvolved simultaneously. This can increase the usage of the GPU for small workloads.

# Chapter 6

# Results

In this chapter we discuss the results obtained by our implementation of OMP. We look at its ability to deconvolve radio interferometric images by deconvolving an image produced using a simulated interferometer and a known sky model. This deconvolved image is compared to the results of the CLEAN algorithm. We also examine the performance characteristics of both our CPU and GPU implementations.

## 6.1 Synthetic Testing

One of the major problems in testing imaging systems for radio astronomy is that there is no completely known astronomical image with which to verify the system. As such, we create a synthetic sky model, which is then run through a simulated KAT-7 interferometer. By knowing the exact sky model, we can easily create a perfect image against which to test.

In order to compare the resulting image to a sky model, we need to extract the candidate sources from the deconvolved image. To do this, we run the images through the *Python Blob Detection and Source Measurement* (PyBDSM) source detection software. This results in a list of candidate sources, which we still need to compare to our sky sources. This provides a residual RMS, $\sigma$, that is, the RMS of the remaining image after the sources have been extracted.

While it is sometimes possible to detect structures smaller than the restoring beam (which are called super-resolution methods), such methods often require the observed image to fit particular constraints. We define that two sources are *nearby* to each other if their restoring beams overlap. In particular, we require that the *Full Width at Half Maximum* (FWHM) of the restoring beams overlap. We can then create a series of rules by which sources are matched:

- First, we consider sources (in the sky model) that are unresolvable due to a nearby, brighter source (also in sky model). These sources are flagged and will not be considered as matched or missing.
- We consider a source (from the sky model) as successfully matched if there is an extracted source (from the deconvolved image) nearby whose intensity (in Jy/beam) differs by no more than $3\sigma$ (which was chosen experimentally, and found to give good results for both

CLEAN and OMP). These sources are then flagged as matched. Any unflagged sources (from the sky model) are then considered to be a missing (a type II error).

- We also consider an extracted source (from the deconvolved image) as spurious (a type I error) if there is no nearby source whose intensity (in Jy/beam) differs by no more than $3\sigma$.
- Since OMP and CLEAN have a different $\sigma$, we also consider the case where the OMP sources are matched to within a difference of CLEAN's $3\sigma$.

In order to ensure that the results are statistically viable, we repeat this process on 100 different sky models.

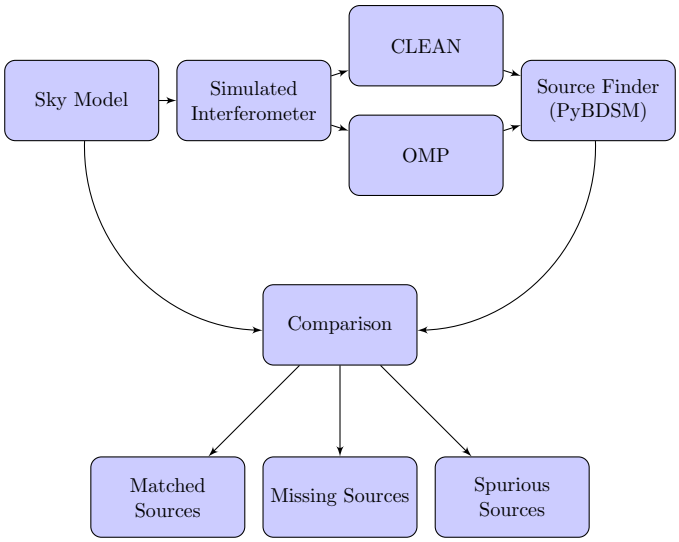



Figure 6.1.1: A simplified diagrammatic overview of the synthetic testing. The sky model is parsed into a simulated interferometer, which produces the dirty image and PSF for that interferometer. Deconvolution is then performed using both CLEAN and OMP, with each producing a deconvolved image. Each of these images is then run through a source finder, and the results are compared to the initial sky model. This comparison will tell us how each deconvolution implementation performed.

### 6.1.1 Comparison to CLEAN

In order to appropriately test OMP's performance for deconvolution, we will deconvolve synthetic images using both OMP and CASA's Högbom CLEAN, and compare the results.

Additionally, we are interested in the statistical properties of the residual image after deconvolution and source extraction.

Visual inspection of a single image is performed to ensure that the algorithms are operating reasonably, and to get an initial idea of what results to expect. Figure 6.1.2 shows the resulting images. It is clear that many of the fainter sources from the sky model, (a), are more easily identifiable in the OMP image, (b), than in the CLEAN image, (d). In addition we can see that far less structure remains after deconvolution by OMP, (c), than after CLEAN, (e).

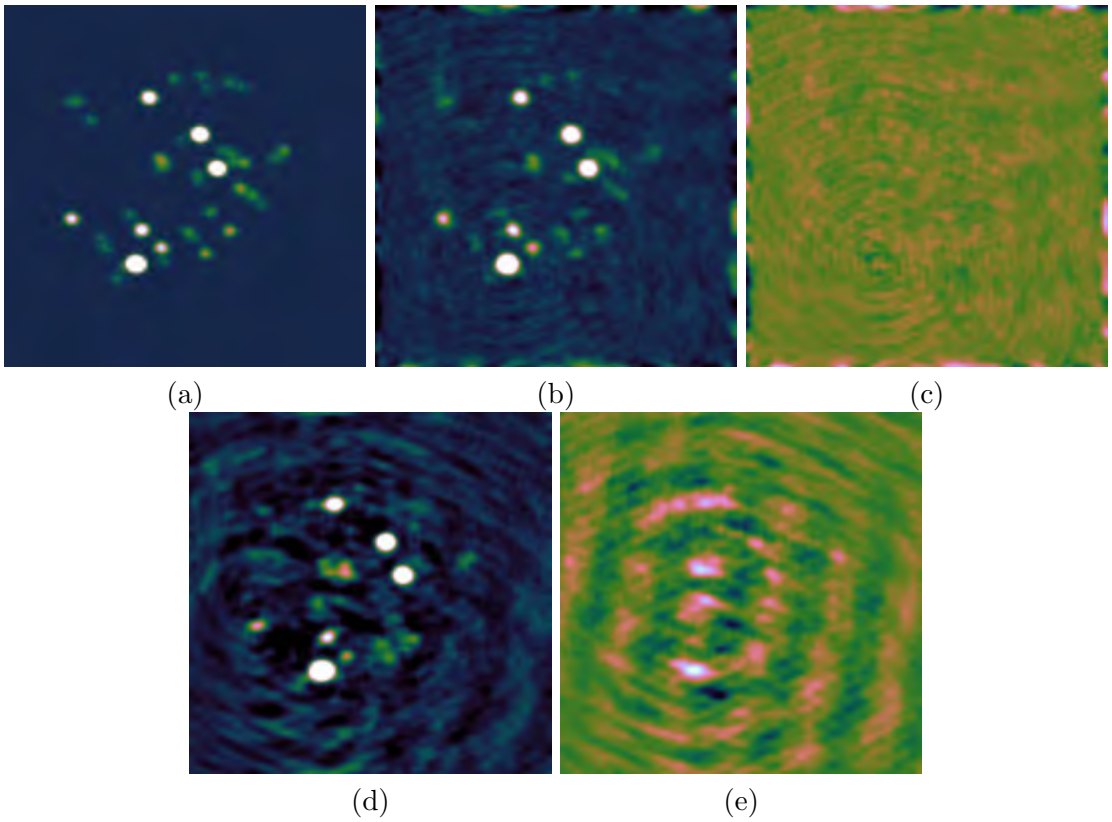



Figure 6.1.2: Deconvolution of the synthesised sky model: (a) the sky model convolved with the restoring beam. This is what an ideal interferometer and deconvolution algorithm would produce. (b) the deconvolved image using OMP. (c) the residual after deconvolution using OMP. (d) the deconvolved image using CLEAN. (e) the residual after deconvolution using CLEAN. Images (a), (b), and (d) are scaled to exclude the peak percentile from image (b), while images (c) and (e) are scaled to the full range of image (e).

The residual is significantly higher around the edges in images (b) and (c). This is likely due to gridding artefacts resulting in the dirty image not being a perfect convolution of the true sky. These can either be ignored (which may result in some low intensity spurious sources), or they can be removed by initially creating a larger image, which is then cropped after deconvolution. They can also be removed by re-gridding the deconvolved image, as is done in CLEAN, at a significant performance cost.

It should be noted that, for this image, the $\ell_2$ norm (which is often used for image comparisons) produced similar results for CLEAN and OMP, with CLEAN slightly outperforming OMP. From looking at the images, however, the results of OMP appear to match the sky model far better than those of CLEAN, with less structure remaining in the residual. As such, we should consider that the $\ell_2$ norm might not be a good way to compare interferometric images of point sources.

A better approach is as follows: 100 synthetic sky models are used to generate dirty images which are then deconvolved using both CLEAN and OMP.

After performing source extraction on both the CLEAN and OMP images and using the comparison described in Section 6.1, we get the results shown in Table 6.1.

OMP significantly outperformed CLEAN, matching 70% (with a standard deviation across all images of $\pm 10\%$) of the 2660 sources contained in all 100 sky models, while CLEAN only matched 45% ($\pm 10\%$) with almost twice an many missing sources. In addition, the residual RMS noise after sources extraction was 2.7 ($\pm 0.8$) times lower for OMP than for CLEAN.

However, OMP produced almost 5 times the number of spurious sources. A disproportionate number of these spurious sources lie in the border regions where OMP has an increased residual. By cropping out this region, we reduce this number somewhat (down to 224 spurious sources from 261). This also eliminates three of the spurious sources produced by CLEAN.

### 6.1.2 Thresholding OMP and CLEAN equally

Since OMP results in a significantly lower residual, it also has a lower threshold for source extraction. As such, the feature might result in an extracted source on the OMP image, but not on the CLEAN image. Thus an important question is whether these extra matched sources, as well as the spurious sources, are purely a result of the lower residual.

As such, a second test was performed by first filtering out any sources below $5\times$ CLEAN's residual. This is around the level chosen for PyBDSM to consider a feature as valid on the CLEAN images. These results are shown in Table 6.2.

In this case, OMP still outperformed CLEAN by around the same margin, matching 76% ($\pm 11\%$) of the 1890 sources compared to CLEAN's 56% ($\pm 10\%$). In addition, this still outperforms the unthresholded CLEAN algorithm, matching 54% of the initial 2660 sources.

In addition, this also eliminates most of the spurious sources outside the border regions.

### 6.1.3 Extracted source accuracy

For this testing, we consider a source to have been matched if its intensity is within a hard threshold, and if the identified position is within a specified distance from the true source. For some applications, it might be important to know how closely the sources resulting from a deconvolution method would match the actual sources.

As such, after matching the sources to the sky model, we performed a test to determine how closely the deconvolved source matches the source in the sky model in both intensity (measured in Jy/beam and in $\sigma$s) and in distance (measured in restoring beam widths). Given the definitions of the residual RMS and resolving power, we expected the implementations to vary in intensity by about $1\sigma$ and in distance by about one beam width.

OMP produced excellent results, with an intensity difference of 0.0083 (with a standard deviation across all matched sources of $\pm 0.0078$) Jy/beam, or $0.75\sigma$ ($\pm 0.61\sigma$) and a distance of $0.057\sigma$ ($\pm 0.103\sigma$) beam widths.

CLEAN, on the other hand, performed poorly when measuring the intensity, with a difference of about $0.040 \pm 0.024$ Jy/beam, or $1.57\sigma \pm 1.06\sigma$, while performing excellently when measuring the distance, $0.068 \pm 0.075$ beam widths.

### 6.1.4 Two-to-one source matching

OMP (and, to a lesser extent, CLEAN) will, where possible, optimise sparsity. This will often result in two unresolvable sources in the sky model being represented by a single source in the deconvolved model. In this case, the resulting deconvolved source will have an intensity close to the sum of the two sources, resulting in it being flagged as a false positive, and the two sources in the sky model flagged as unmatched sources.

As such, we modified the source-matching algorithm to consider two sources a match if their intensities are within a factor of two. This will allow for the possibility that up to two unresolvable sources are deconvolved into a single source, or that a single source is deconvolved into two unresolvable sources.

In this case, OMP now outperforms CLEAN by a larger margin, matching 82% ($\pm 9\%$) of the 2660 sources compared to CLEAN's 61% ($\pm 11\%$). In particular, both algorithms now produce far fewer spurious sources, with CLEAN producing only 10 (7 of which lie outside the border region), and OMP producing 50, only 14 of which lie outside the border region.

If we again filter out sources below $5\times$ CLEAN's threshold, OMP matches 92% ($\pm 8\%$) of the 1890 sources compared to CLEAN's 84% ($\pm 9\%$). OMP now only produces slightly more spurious sources than CLEAN, producing 7 spurious sources (5 of which lie outside the border region), compared to CLEAN's 5 spurious sources (4 of which lie outside the border region).

If we also loosen the beam width constraints (such that we consider two sources a match if they are within 1.5 beam widths, instead of 1 beam width), OMP will produce fewer spurious sources than CLEAN. This implies that, for the most part, there is at least a feature near the spurious sources produced by OMP.

| | CLEAN | OMP | OMP Relaxed |
|---|---|---|---|
| Masked Sources | 378 | 378 | 378 |
| Matched Sources | 1185 | 1837 | 1979 |
| Missing Sources | 1475 | 823 | 681 |
| Spurious Sources | 56 | 261 | 169 |
| Spurious Sources outside borders | 53 | 224 | 133 |
| Residual RMS noise | 0.0274 | 0.0101 | 0.0101 |

Table 6.1: Comparison of CLEAN and OMP. Source counts are summed over all 100 sky models. OMP Relaxed refers to matching sources if they differ by no more than CLEAN's $3\sigma$ instead of OMP's.

| | CLEAN | OMP | OMP Relaxed |
|---|---|---|---|
| Matched Sources | 1165 | 1430 | 1559 |
| Missing Sources | 725 | 460 | 331 |
| Spurious Sources | 51 | 218 | 126 |
| Spurious Sources outside borders | 50 | 215 | 124 |

Table 6.2: Thresholding at CLEAN's $5\sigma$: comparison of CLEAN and OMP. Source counts are summed over all 100 sky models. OMP Relaxed refers to matching sources if they differ by no more than CLEAN's $3\sigma$ instead of OMP's.

| | CLEAN | OMP |
|---|---|---|
| Matched Sources | 1594 | 2139 |
| Missing Sources | 1066 | 521 |
| Spurious Sources | 10 | 50 |
| Spurious Sources outside borders | 7 | 14 |

Table 6.3: Allowing for two-to-one source matching: comparison of CLEAN and OMP. Source counts are summed over all 100 sky models.

| | CLEAN | OMP |
|---|---|---|
| Matched Sources | 1571 | 1717 |
| Missing Sources | 319 | 173 |
| Spurious Sources | 5 | 7 |
| Spurious Sources outside borders | 4 | 5 |

Table 6.4: Allowing for two-to-one source matching and as thresholding at CLEAN's $5\sigma$: comparison of CLEAN and OMP. Source counts are summed over all 100 sky models.

## 6.2 Runtime Performance

Due to the increasing resolution of interferometers and, in particular, the SKA's large leap in required image size, it is important to accelerate all parts of the interferometry pipeline.

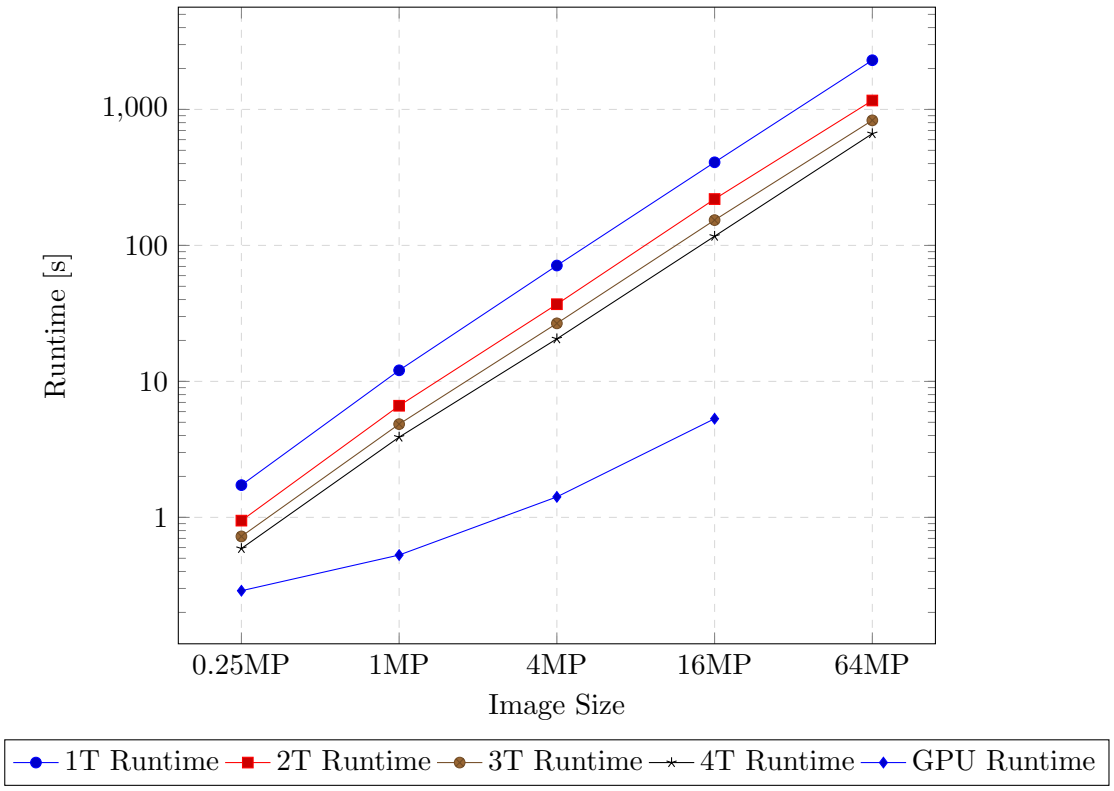



Figure 6.2.1: Runtime for 100 iterations on various image sizes using an Intel i7-4770 and a NVIDIA GTX 770. Note that the axes are logarithmic.

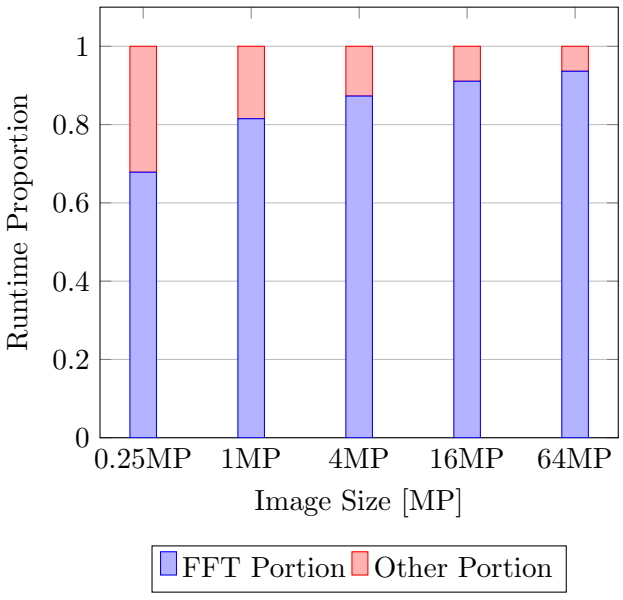



Figure 6.2.2: Single Threaded execution time breakdown (after 100 iterations).

Furthermore, near real-time image synthesis allows for more experimentation on the part of the astronomer. Thus, speedup measures form an important part of evaluating a system.

One important consideration is the floating point precision of the program. In particular, Kepler GPUs support two levels of precision, a 32-bit *single precision* floating point type (float), and a 64-bit *double precision* floating point type (double). Since the effect of precision was not known beforehand, both options were implemented. However, on the test dataset, the difference between the resulting image under the two precision types was negligible. As such, single precision was chosen as it executes significantly faster.

Figure 6.2.1 shows the base runtime for a single-threaded execution of the implementation for 100 iterations on various image sizes. We can see that the 0.25MP and 1MP image take considerably less than a minute to complete execution, with the 4MP image only requiring slightly more than a minute. The 16MP and 64MP images take much longer, with the 64MP image requiring about 38 minutes to complete.

In order to improve on this performance, we can make use of multi-core CPUs. As we can see from Figure 6.2.3, a 4-core CPU achieves around a 3.5 times speedup for a sufficiently large image. There is no GPU datapoint for 64MP due to the limited GPU memory of the test system (2GB).

The speedup for smaller images (smaller than 4MP) is only relevant if a large number of images need to be processed at once, as their execution time is already sufficiently quick. However, in the case of a large number of simultaneous images, the images can be deconvolved in parallel, allowing for even greater parallelism.

Furthermore, as is shown in Figure 6.2.4, with a NVIDIA GTX770, we can obtain up to a 83 times speedup for a 16MP image over the single-threaded CPU implementation. This means that a 16MP image on a GPU only requires around 5 seconds to deconvolve, compared to the

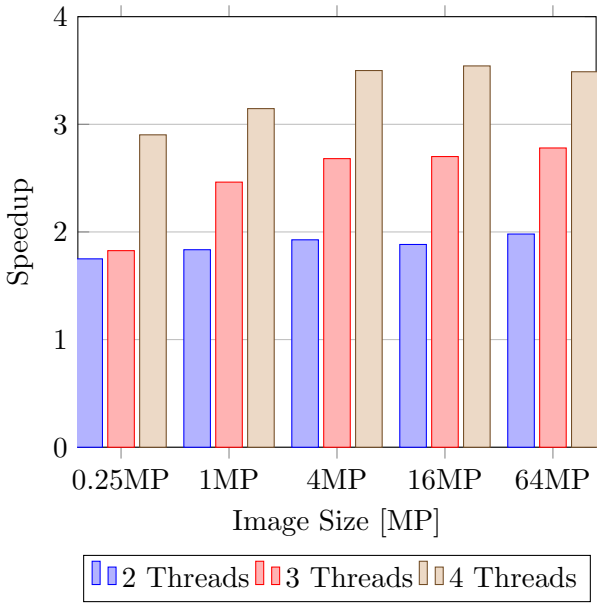



Figure 6.2.3: Speedup obtained on a 4-core CPU (Intel Core i7-4770) for 2, 3 and 4 threads.

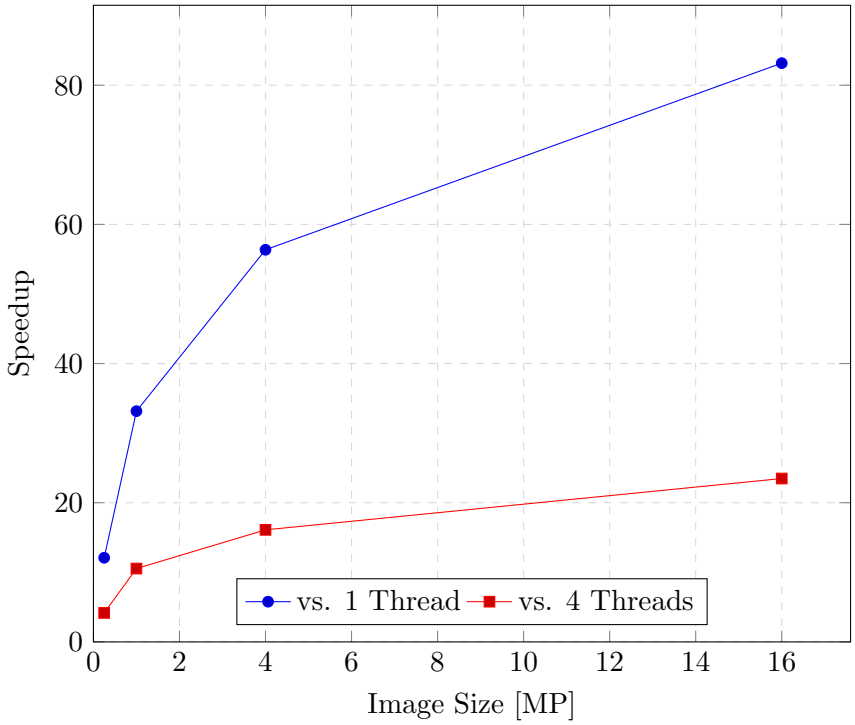



Figure 6.2.4: Speedup obtained on a GTX 770, compared to the single-threaded and 4-threaded CPU implementation running on an Intel Core i7-4770.

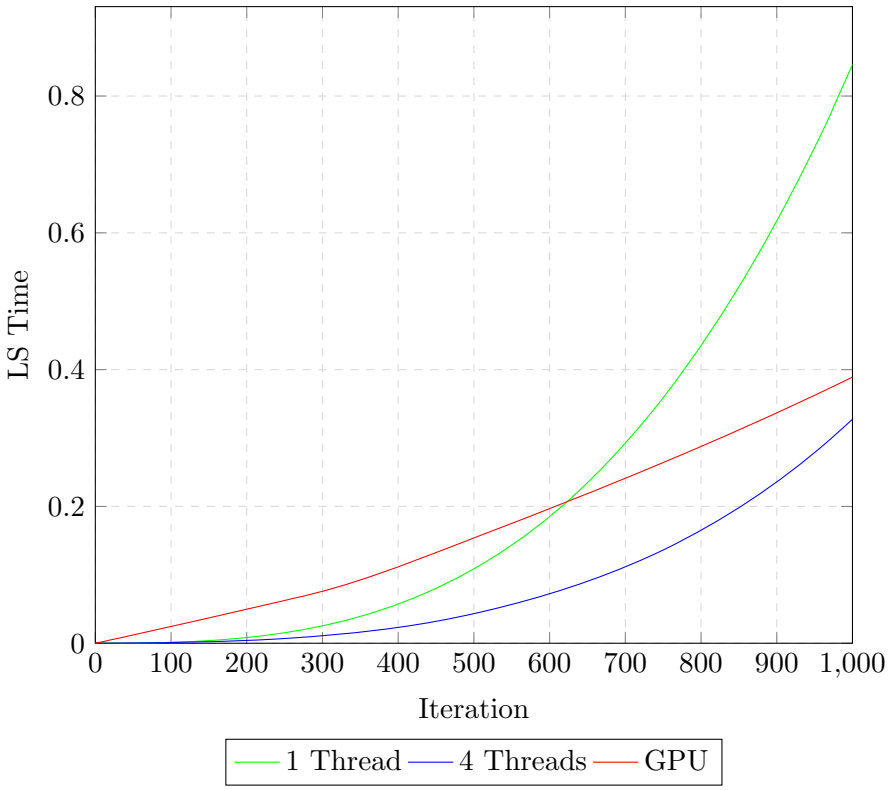



Figure 6.2.5: Time taken to re-weight the selected pixels (matrix inverse update), as a proportion of the total runtime for that implementation.

CPU's 400 seconds.

Figure 6.2.5 shows the time spent calculating the inverse matrix. Initially, the inverse matrix is small enough that the update requires almost no work. As such the GPU implementation initially suffers due to the relatively large kernel execution overhead. However, as the workload increases, this overhead becomes less significant, and the GPU implementation exhibits slower growth than the CPU implementation.

Since the time taken to compute the inverse matrix is independent of the image size, its contribution to the total runtime becomes insignificant for larger images. When deconvolving a 16MP image, this contribution is less than 1% after 1000 iterations.

In conclusion, we find that OMP extracts significantly more sources than CLEAN (up to 82% compared to 61% over 100 synthetic sky models), and having a significantly lower residual (2.7 times lower than that of CLEAN). However, it also produces more spurious sources than CLEAN, though these can be somewhat mitigated.

In addition, the GPU implementation of OMP provides a significant speedup over the CPU implementation, achieving an 83× speedup over the single-threaded implementation, and a 23× speedup oved the 4-threaded CPU implementation.

# Chapter 7

# Conclusion

Observing the radio sky with a radio interferometer results in an image of the sky that is convolved with the point spread function. Deconvolution algorithms attempt to recover the true sky from this convolved image. Radio astronomers typically use a variant of the CLEAN algorithm for this purpose.

Based on the assumption of a sparse sky model, we instead consider Compressed Sensing techniques for deconvolution. In particular, we adapt the Orthogonal Matching Pursuit (OMP) algorithm to deconvolution.

Since a radio interferometer measures points on the 2-dimensional Fourier plane, we can adapt OMP to use Fast Fourier Transforms (FFTs) instead of Direct Fourier Transform matrices. This allows us to reduce the complexity of OMP, which allows for a feasible runtime for standard image sizes.

In order to evaluate the effectiveness of OMP for deconvolution, we generate synthetic sky models and run them through a simulated interferometer. We can then deconvolve the resulting images with both OMP and CLEAN, and compare the results to the initial sky model.

In terms of image quality, we find that OMP extracts significantly more sources than CLEAN, extracting up to 82% of the sources over the 100 sky models, compared to CLEAN's 61%. In addition, the residual after source extraction is 2.7 times lower for OMP than for CLEAN.

For OMP, the residual is elevated around the edges of the image, resulting in occasional spurious sources in this region. As such, a wider image should be used, which can be cropped after deconvolution. Due to the lower residuals, there are also some low-intensity spurious sources which would be lost in the noise of the CLEAN image.

Since future radio interferometers, the Square Kilometre Array in particular, expect to have significantly larger images to deconvolve. Deconvolution of these larger images thus requires faster deconvolution implementations.

As such, we adapt our OMP implementation to make use of parallel architectures. In particular, we create a multi-core CPU implementation using OpenMP, and a GPU implementation using nVidia's CUDA.

Our GPU implementation achieves a 83× speedup over the single-threaded implementation, and a 23× speedup over the 4-threaded CPU implementation (that is, our parallel CPU implementation achieves a near-linear speedup at 4 CPU cores).

## 7.1 Future Work

OMP, like CLEAN, is designed to deconvolve point sources. However, CLEAN can still prove surprisingly effective at resolving extended structures. It would be beneficial to evaluate how OMP performs when tasked with deconvolving an extended emission.

Additionally, there are several variants of CLEAN that increase its effectiveness when deconvolving extended structure transforming the image into a sparser basis, such as multi-scale CLEAN or wavelet CLEAN. In this, a transform of the PSF is subtracted from the transform of the dirty image to produce a transform of the deconvolved image.

A similar approach could prove beneficial to OMP for deconvolving extended structure, however the algorithmic optimisations will need to be reworked using the new sensing matrix. In particular, some of the optimisations might become impossible under some transforms (particularly those which cannot be expressed as a matrix transform), which would result in infeasible execution times.

OMP perfectly optimises the intensity of all selected sources in each iteration. However, bad positioning of a source might never be corrected throughout the algorithm execution. It might further reduce the residual if an OMP algorithm was developed which also optimised the source position. This might be achieved by also adding several pixels near each source when performing the least-squares optimisation. The pixel with the maximum resulting intensity should be the best candidate for a source.

Additionally, if the maximal intensities diverge in two or more directions from the initial source, this could indicate that multiple sources are within the resolving ability of the interferometer, and could be a method to achieve super-resolution. Alternatively, this could be set off by noise, resulting in many more spurious sources and reducing the image sparsity.

The constant improvement of technology, particularly graphics hardware, makes writing future-proof high-performing software a challenge. In order to combat this, our OMP implementation abstracts the specific hardware by using libraries likely to be frequently updated to make use of the latest hardware optimisations. Thus, so long as the library interfaces remain consistent, this implementation should retain good performance on future hardware. As such, this implementation should be tested on newer hardware and larger data sets in the future.

# Chapter 8

# References


Greg Baker, John Gunnels, Greg Morrow, Beatrice Riviere, and Robert van de Geijn. Plapack: High performance through high-level abstraction. In *Parallel Processing, 1998. Proceedings. 1998 International Conference on*, pages 414–422. IEEE, 1998.

Emmanuel Candes and Justin Romberg. Sparsity and incoherence in compressive sampling. *Inverse problems*, 23(3):969, 2007.

Emmanuel J Candès. Compressive sampling. In *Proceedings on the International Congress of Mathematicians: Madrid, August 22-30, 2006: invited lectures*, pages 1433–1452, 2006.

RE Carrillo, JD McEwen, and Yves Wiaux. Sparsity averaging reweighted analysis (sara): a novel algorithm for radio-interferometric imaging. *Monthly Notices of the Royal Astronomical Society*, 426(2):1223–1234, 2012.

BG Clark. An efficient implementation of the algorithm 'CLEAN'. *Astronomy and Astrophysics*, 89:377, 1980.

JE Conway, TJ Cornwell, and PN Wilkinson. Multi-frequency synthesis-a new technique in radio interferometric imaging. *Monthly Notices of the Royal Astronomical Society*, 246:490, 1990.

Tim J Cornwell. Multiscale CLEAN deconvolution of radio synthesis images. *Selected Topics in Signal Processing, IEEE Journal of*, 2(5):793–801, 2008.

Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

Yong Fang, Liang Chen, Jiaji Wu, and Bormin Huang. Gpu implementation of orthogonal matching pursuit for compressive sensing. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 1044–1047. IEEE, 2011.

Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137. ACM, 2004.

Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

Mike Garrett. Calibration and advanced radio interferometry, 2010. URL `https://www.astron.nl/~mag/dokuwiki/lib/exe/fetch.php?media=radio_astronomy_lec_8_ma_garrett.pdf`.

Hugh Garsden, JN Girard, Jean-Luc Starck, Stéphane Corbel, C Tasse, A Woiselle, JP McKean, Alexander S Van Amesfoort, J Anderson, IM Avruch, et al. Lofar sparse image reconstruction. *Astronomy & astrophysics*, 575:A90, 2015.

Mark Harris. Unified memory in cuda 6, 2013. URL `devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6`.

Jared Hoberock and Nathan Bell. Thrust: C++ template library for cuda, 2009. URL `code.google.com/p/thrust`.

JA Högbom. Aperture synthesis with a non-regular distribution of interferometer baselines. *Astronomy and Astrophysics Supplement Series*, 15:417, 1974.

S Kazemi and S Yatawatta. Robust radio interferometric calibration using the t-distribution. *Monthly Notices of the Royal Astronomical Society*, 435(1):597–605, 2013.

Stefan Kunis and Holger Rauhut. Random sampling of sparse trigonometric polynomials, ii. orthogonal matching pursuit versus basis pursuit. *Foundations of Computational Mathematics*, 8(6):737–763, 2008.

HJ Landau. Necessary density conditions for sampling and interpolation of certain entire functions. *Acta Mathematica*, 117(1):37–52, 1967.

André Lannes, Eric Anterrieu, and Pierre Maréchal. Clean and wipe. *Astronomy and Astrophysics Supplement Series*, 123(1):183–198, 1997.

Charles L Lawson and Richard J Hanson. *Solving least squares problems*, volume 161. SIAM, 1974.

Michael Lustig, David Donoho, and John M Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic resonance in medicine*, 58(6):1182–1195, 2007.

Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

R Narayan and R Nityananda. Indirect imaging. In *Proc. IAU/URSI Symp., ed. JA Roberts, Cambridge Univ. Press*, page 267, 1984.

CUDA Nvidia. Cublas library, 2008. URL `developer.nvidia.com/cuBLAS`.

CUDA Nvidia. Cufft library, 2010. URL developer.nvidia.com/cuFFT.

*NVIDIA GeForce GTX 980 Whitepaper 1.1*. NVIDIA Corporation, 2014a. URL http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF.

*CUDA C Programming Guide 6.5*. NVIDIA Corporation, 2014b. URL http://docs.nvidia.com/cuda/cuda-c-programming-guide.

*Cuda Parallel Thread Execution ISA Version 4.1 (v6.5)*. NVIDIA Corporation, 2014c. URL http://docs.nvidia.com/cuda/cuda-c-programming-guide.

Urvashi Rau and Tim J Cornwell. A multi-scale multi-frequency deconvolution algorithm for synthesis imaging in radio interferometry. *Astronomy & Astrophysics*, 532:A71, 2011.

FR Schwab. Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry. *The Astronomical Journal*, 89:1076–1081, 1984.

Ludwig C Schwardt. Compressed sensing imaging with the kat-7 array. In *Electromagnetics in Advanced Applications (ICEAA), 2012 International Conference on*, pages 690–693. IEEE, 2012.

Avi Septimus and Raphael Steinberg. Compressive sampling hardware reconstruction. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 3316–3319. IEEE, 2010.

Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37 (1):10–21, 1949.

JL Starck, E Pantin, and F Murtagh. Deconvolution in astronomy: A review. *Publications of the Astronomical Society of the Pacific*, 114(800):1051–1069, 2002.

Greg B Taylor, Chris Luke Carilli, and Richard A Perley. Synthesis imaging in radio astronomy ii. In *Synthesis Imaging in Radio Astronomy II*, volume 180, 1999.

AR Thompson and RN Bracewell. Interpolation and fourier transformation of fringe visibilities. *The Astronomical Journal*, 79:11–24, 1974.

Josep Torrellas, Monica S. Lam, and John L Hennessy. False sharing and spatial locality in multiprocessor caches. *Computers, IEEE Transactions on*, 43(6):651–663, 1994.

Joel Tropp, Anna C Gilbert, et al. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.

Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. Intel math kernel library. In *High-Performance Computing on the Intel Xeon Phi*, pages 167–188. Springer, 2014.

Yves Wiaux, Laurent Jacques, Gilles Puy, AMM Scaife, and Pierre Vandergheynst. Compressed sensing imaging techniques for radio interferometry. *Monthly Notices of the Royal Astronomical Society*, 395(3):1733–1742, 2009.

# Appendices

# Appendix A

# Algorithm progression snapshots

This appendix provides snapshots at various points in the algorithm progression of a single sky image. These snapshots consist of a restored sky model at the current iteration, with the residue at that iteration added in. This allows an observer to perceive the change in the background with each iteration. All these images use the same color map and scale.

This particular image is considered optimally deconvolved after around 30 iterations. Any further iterations produce little change in the residue, are unlikely to find any more sources from the sky model, and start to produce spurious sources.

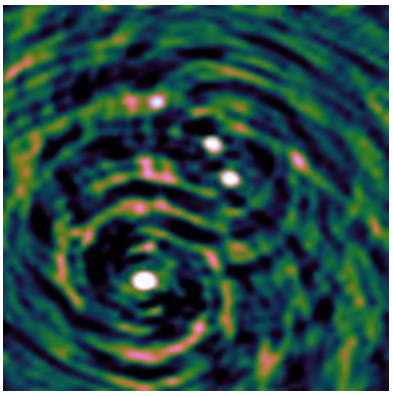


Figure A.1: Iteration 0

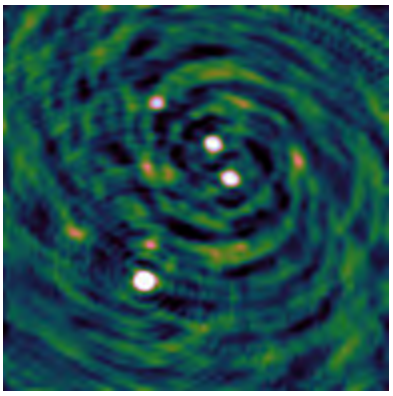


Figure A.2: Iteration 1

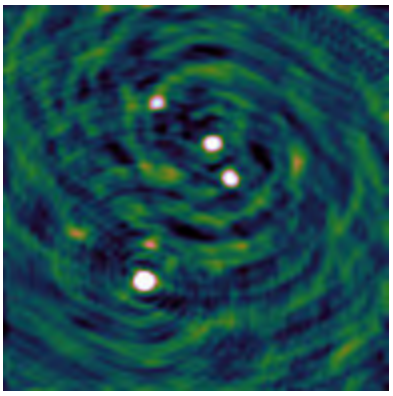


Figure A.3: Iteration 2

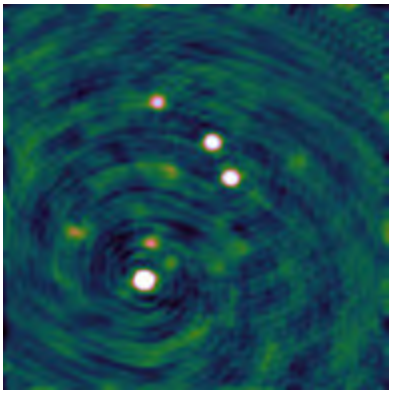


Figure A.4: Iteration 4

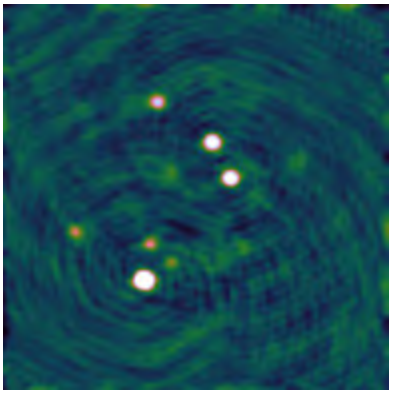


Figure A.5: Iteration 8

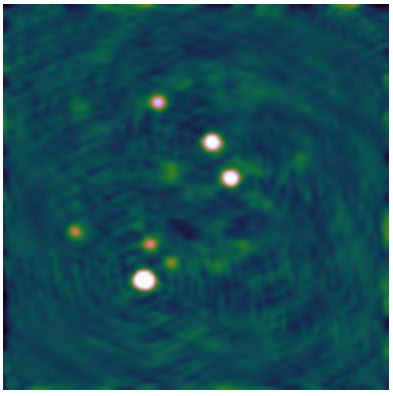


Figure A.6: Iteration 16

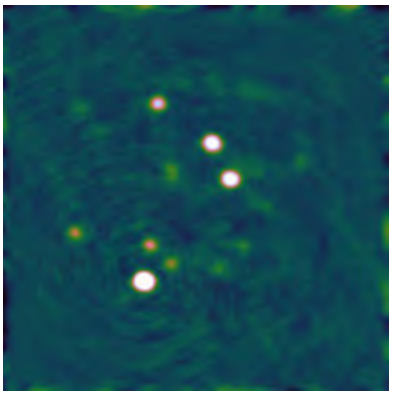


Figure A.7: Iteration 32

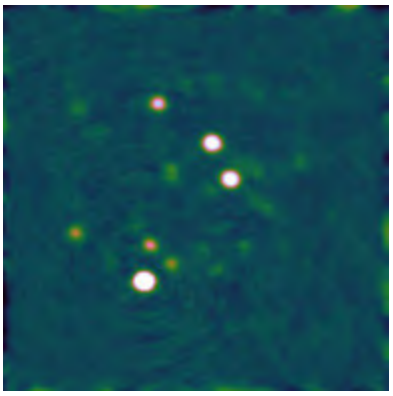


Figure A.8: Iteration 64

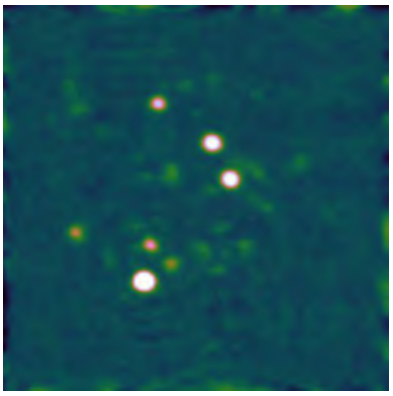


Figure A.9: Iteration 128